Wave Analytics Dashboard JSON Guide

Salesforce, Spring '16

'16

CONTENTS

WAVE ANALYTICS DASHBOARD JSON OVERVIEW

Create advanced dashboards in Wave Analytics by modifying the JSON that defines a dashboard. The easiest way to design dashboards is to use the designer. However, you can further customize dashboards by editing their JSON files.

Modify a dashboard's JSON file for tasks such as:

- Specify a SAQL query, and specify relationships between the query and other steps.
- Populate a selector with a specified list of values instead of from a query.
- Use manual bindings to override the default faceting and manually specify the relationships between the steps.
- Set query limits.
- Specify columns for a values table.
- Set up layouts for mobile devices.

> **Note:** Wave provides two dashboard designers. Except for the gridLayouts content, this guide discusses how to perform tasks in the original designer. The `gridLayouts` section discusses how to customize the JSON when using the new flex dashboard designer, which is beta as of Spring '16.

VIEW OR MODIFY A DASHBOARD JSON FILE

To create advanced dashboards, you typically modify the JSON file that defines a dashboard.

Expert Editor Mode provides the JSON of a lens or dashboard and lets you quickly see the effect of your edits in the running asset.

> **Note:** Expert Editor Mode replaces the tool at
> `https://`**`your_Salesforce_instance`**`/insights/web/lens.apexp`. For lenses and dashboards opened, created, or saved in the Spring '16 release, use the JSON editor instead of the `lens.apexp` page. A lens or dashboard from a previous release can be edited in either place, but after saving in the Spring '16 release, it no longer appears on the `lens.apexp` page.

1. To access Expert Editor Mode, open the lens or dashboard you want to edit, and press CTRL+E for PCs or CMD+E for Macs.

2. If the **Reload JSON** button is available, click it to load the current running lens or dashboard. **Reload JSON** loads the code of the currently running lens or dashboard. This button is available whenever the JSON in the editor doesn't match the running JSON, which can happen when you first open the editor.

3. Modify the JSON in the editor. You can use standard keyboard shortcuts for editing functions and search.

4. To go back to the explorer and see how edits to the JSON appear in the lens or dashboard, click **Switch to Runtime**.

   > **Important:** **Switch to Runtime** overwrites the JSON of the running lens or dashboard with the JSON in the editor.

5. To retain your edits, save the lens or dashboard. Changes made in the JSON editor are not saved until you explicitly save the lens or dashboard.

In Expert Editor Mode, the following shortcuts let you perform basic actions from your keyboard.

Expert Editor Mode Keyboard Shortcut	Description
CRTL+3 (Windows); CMD+3 (Mac)	Load JSON from runtime
CRTL+X (Windows); CMD+X (Mac)	Cut
CRTL+C (Windows); CMD+C (Mac)	Copy
CRTL+V (Windows); CMD+V (Mac)	Paste
CRTL+Z (Windows); CMD+Z (Mac)	Undo
SHIFT+CRTL+Z (Windows); SHIFT+CMD+Z (Mac)	Redo
CRTL+F (Windows); CMD+F (Mac)	Search (RegExp, case-sensitive, or whole word searches available)
CRTL+E (Windows); CMD+E (Mac)	Switch to runtime

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To modify the JSON file that defines a dashboard:
- "Create and Edit Wave Analytics Dashboards"

DASHBOARD JSON FILE EXAMPLE

A dashboard JSON file defines the components that a dashboard contains and describes how they're connected.

This sample JSON file defines a simple dashboard that uses a chart widget to display the count of rows in a dataset, grouped by a field (account owner). It includes one lens, called "AccountOwner_Owner_Name_6," and one widget, called "chart_7." The `datasets` section lists the dataset that the dashboard uses. The `layouts` section specifies a mobile layout with one page, one row, and one column.

```json
{
    "label":"Simple Dashboard",
    "state":{
        "steps":{
            "AccountOwner_Owner_Name_6":{
                "isFacet":true,
                "isGlobal":false,
                "query":{
                    "measures":[["count","*"]],
                    "groups":["AccountOwner.Owner.Name"]},
                "selectMode":"single",
                "type":"aggregate",
                "useGlobal":true,
                "start":null,
                "datasets":[{"name":"Opportunities"}],
                "visualizationParameters":{"visualizationType":"hbar"}}},
        "widgets":{
            "chart_7":{
                "type":"chart",
                "position":{"zIndex":6,"x":0,"y":20,"w":"410","h":"300"},
                "parameters":{
                    "step":"AccountOwner_Owner_Name_6",
                    "visualizationType":"hbar"}}},
    "datasets":[{
            "id":"0FbR000000000O3KAI",
            "name":"Opportunities"}]}
}
```

STEPS

The `steps` section contains the queries that you've clipped from the explorer.

Each step has a name that's used to link it to a widget that's defined elsewhere in the JSON file.

The properties of the `steps` section of a dashboard JSON file are described in the table.

Field Name	Description
`datasets`	An array of datasets used by this step. Specify the alias of each dataset. Currently, only the first dataset is used. ✏️ Note: Faceted steps are filtered based on only the first dataset specified in this array.
`visualizationParameters`	Visualization details about the step. Example: ``` "visualizationParameters": "visualizationType":"hbar" ``` Valid values for visualizationType are: - `calheatmap*`—calendar heat map - `comparisontable`—compare table - `hbar`—horizontal bar - `hdot*`—horizontal dot plot - `heatmap*`—heat map - `matrix*`—matrix - `parallelcoords*`—parallel coordinates - `pie`—donut - `pivottable*`—pivot table - `scatter`—scatter plot - `stackhbar`—stacked horizontal bar - `stackvbar`—stacked vertical bar - `time`—timeline - `valuestable`—raw data table - `vbar`—vertical bar - `vdot*`—vertical dot plot ✏️ Note: The flex dashboard designer doesn't support charts with an asterisk (*). If you specify an unsupported type, the designer replaces it with a horizontal bar (`hbar`) in the dashboard.

Field Name	Description
isFacet	Enables bi-directional faceting between this step and other steps built from the same dataset, which is specified in `datasets` field for this step. Set to `true` or `false`. 📝 Note: If a SAQL query is based on multiple datasets, only the first dataset specified in the `datasets` field is faceted. Also, `isFacet` works only for compact-form queries, by default. To enable them for a SAQL query, also set the `autoFilter` option to `true`.
isGlobal	Indicates whether the filter that's specified in the query is used as a global filter (`true`) or not (`false`). Default is `false`. You can only apply this property on steps that are connected to a scope widget—all other steps ignore this property. A global filter filters other steps in the dashboard that have `useGlobal` set to `true` and reference the same dataset. By default, it filters compact-form steps only. To filter a SAQL step, set `autoFilter` to `true` in the SAQL step.
query	The query that the step uses. It can be in SAQL or compact form.
selectMode	Determines the selection interaction. The options for charts are: `none`, `single`, and `singlerequired`. The options for list and toggle selectors are: `single`, `singlerequired`, `multi`, and `multirequired`. 📝 Note: `selectMode` isn't used by the number, values table, compare table, range, date, and global filter widgets.
start	The default start value or values for a step. This value is used when a dashboard is initialized or refreshed.
type	The type can be set to: • `aggregate`. See an example. • `grain`. See an example. • `multi`. See an example. • `static`. See an example.
useGlobal	Indicates whether the step uses the dashboard's global filter (`true`) or not (`false`).

dimensions	The dimension used to facet other steps. Wave facets other steps based on the value selected for this dimension in the user interface. Specify the `dimensions` attribute only if `isFacet` is set to `true`.
	
	Example:
	<pre>"step_filter_dim": {
	    "type": "static",
	    "dimensions": ["Product"],
	    "datasets":[{"name":"opportunity"}],
	    "selectMode": "single",
	    "values": [
	        {"value": ["EKG Machine"]},
	        {"value": ["Mammography Machine"]},
	        {"value": ["Ultrasound Machine"]}
	    ],
	    "isFacet": true
	},</pre>

Field Name	Description
values	Values used to filter the results of a static step. For example, you can use these values to populate a date selector.

```
"step_date_static_with_start": {
    "type": "static",
    "values": [
        {
            "display": "-6 years",
            "value": [[["year",-6],["year",0]]]
        },
        {
            "display": "-5 years",
            "value": [[["year",-5],["year",0]]]
        },
        {
            "display": "-4 years",
            "value": [[["year",-4],["year",0]]]
        }
    ],
    "selectMode": "singlerequired",
    "start": [[[["year",-5],["year",0]]]]
}
```

You can also populate a selector from a specified list of static values, instead of from a query.

Static Steps

You can also populate a selector from a specified list of static values, instead of from a query.

A static step is shown in this example. This static step is used for filtering, but static steps can also be created for groups, measures, sort order, and limits.

```
"steps": {
  "step_static_1": {
    "type": "static",
    "dimensions": [ "Stages" ],
    "datasets":[{"name":"opp"}],
    "values": [
      {
        "display": "1",
        "value": "1",
        "measure": 100000
      }, {
        "display": "2",
        "value": "2",
        "measure": 200000
      }, {
        "display": "3",
        "value": "3",
        "measure": 300000
      }, {
        "display": "4",
        "value": "4",
        "measure": 400000
      }, {
        "display": "5",
        "value": "5",
        "measure": 500000
      }
    ],
    "selectMode": "single"
  },
```

For more information, see Selection Binding in a Static Step.

WIDGETS

The `widgets` section defines the widgets that appear in the dashboard. Each widget has a name.

The properties of the `widgets` section of a dashboard JSON file are:

Field Name	Description
`parameters`	Widget parameters vary depending on the type of widget. The `step` element defines the step attached to a widget. For detailed information about different parameters, see Widget Parameters Property Reference.

`position`	Specifies the position of the widget in a dashboard created with the original dashboard designer. Position can consist of the following properties:
	
	x and y
	Specifies the top left corner of the widget. The values of these fields must be integers.
	
	w and h
	Specifies the width and height, respectively. You can enter "auto," percentages ("36%"), and integers ("20") as a string value.
	
	zIndex
	Determines the position of a widget relative to other widgets in the dashboard. zIndex specifies whether a widget is in front of or behind another widget. A smaller zIndex means that a widget appears further behind other widgets with larger zIndex values. The value must be an integer.
	
	Example:
	
	``
	"position": {
	  "x": 40,
	  "y": 40,
	  "w": "300",
	  "h": "auto"}
	``
	
	Measurements are in pixels.
	
	📝 Note: The flex dashboard designer ignores these settings and uses the `position` attribute specified under the `gridLayouts` section of the dashboard JSON.

`type`	The widget type specifies one of the other supported widget types. The value of this field must be a string. • `box`—available in the original dashboard designer only • `chart` • `comparetable` • `container`—available in the flex dashboard designer only • `dateselector` • `globalfilters` • `link` • `listselector` • `number`

Field Name	Description

- `pillbox`
- `rangeselector`
- `text`
- `url`—available in the original dashboard designer only
- `valuestable`

> **Note:** The flex dashboard designer doesn't support box and url widgets. If you open a dashboard in flex designer, the designer removes these widget types from the dashboard. Also, the original dashboard designer doesn't support the container widget—use a box widget instead.

Widget Parameters Property Reference

The `parameters` property of the `widgets` section defines the attributes of a widget in a dashboard. Each widget has its own `parameters` property.

SEE ALSO:

Widget Parameters Property Reference

Widget Parameters Property Reference

The `parameters` property of the `widgets` section defines the attributes of a widget in a dashboard. Each widget has its own `parameters` property.

The parameters available for each widget depend on the widget's `type` property. For example, a `chart` widget can have the `legend` parameter, but a `text` widget can't.

Some parameters are exposed and editable in the dashboard designer's user interface as widget properties. Others are only editable via JSON.

This example excerpt from a dashboard JSON file describes a dashboard with a single `chart` widget. The `chart` widget has four parameters set: `miniBars`, `visualizationType`, `sqrt`, and `step`.

```
"widgets": {
  "chart_1": {
    "parameters": {
      "miniBars": 14,
      "visualizationType": "vbar",
      "sqrt": true,
      "step": "Customer_Name_1"
    },
    "type": "chart",
    "position": {
      "w": "1000",
      "h": "500",
      "zIndex": 0,
      "x": 20,
      "y": 20
    }
```

```
    }
}
```

The widget properties set by the `parameters` property are:

Property Name	Details
backgroundColor	**Type** String **Available for This Widget** • In original designer: `box` • In flex dashboard designer: all widgets, except `chart`, `number`, and `pillbox` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The color of the background. Specify the color in this format: `rgb(`*a, b, c, d*`)`. Using a number between zero and 255, *a* indicates how much red is in the color, *b* how much green, and *c* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, *d* indicates the level of transparency. A value of `0` is invisible and `1` is opaque. For example, `rgb(0, 0, 0, 0.93)` sets the color to a nearly opaque black. `rgb(255, 0, 0, 0.14)` sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. `#000000` indicates black in hexadecimal. `#ff0000` indicates red. The default value is `rgba(0, 140, 201, 0.35)`.
borderColor	**Type** String **Available for This Widget** • In original designer: `box` • In flex dashboard designer: all widgets, except `chart`, `number`, and `pillbox` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The color of the border. Specify the color in this format: `rgb(`*a, b, c, d*`)`. Using a number between zero and 255, *a* indicates how much red is in the color, *b* how much green, and *c* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color.

Property Name	Details
	Using a number between zero and one, *d* indicates the level of transparency. A value of `0` is invisible and `1` is opaque. For example, `rgb(0, 0, 0, 0.93)` sets the color to a nearly opaque black. `rgb(255, 0, 0, 0.14)` sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. `#000000` indicates black in hexadecimal. `#ff0000` indicates red. The default value is `rgba(0,0,0,0.25)`.
`borderRadius`	**Type** Integer **Available for This Widget** • All widgets in flex dashboard designer, except `chart`, `number`, and `pillbox` **Exposed in the Dashboard Designer's User Interface** No. Only editable via JSON. **Description** The roundness of the border corners. Valid values are: `0`, `4`, `8`, and `16`. The higher the value, the more rounded the corner. The default value is `0` (not rounded).
`borderWidth`	**Type** Integer **Available for This Widget** • All widgets in flex dashboard designer, except `chart`, `number`, and `pillbox` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The thickness of the border. Valid values are: `1`, `2`, `4`, and `8`. The higher the value, the thicker the border. The default value is `2`.
`compact`	**Type** Boolean **Available for These Widgets** • `listselector` • `number` • `pillbox` **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether displayed numbers are abbreviated (`true`) or not (`false`).

Property Name	Details
	For example, if `true`, the number 48,081 appears as 48k. Although the number appears to be rounded, it is not. The value 48,081 is preserved when performing mathematics and in charts. If `false`, then 48,081 appears as 48,081. The default value is `false`.
`computeTotal`	**Type** Boolean **Available for These Widgets** • `chart` (only when `visualizationType` is `stackwaterfall` and `waterfall`) **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether to include the total measure column (`true`) or not (`false`). The default value is `true`.

`containedWidgets`	**Type**
	List
	
	Available for This Widget
	
	• `container`
	
	Exposed in the Dashboard Designer's User Interface
	Yes
	
	Description
	An array of all widgets inside the container widget.
	
	Example
	This example shows two widgets (`meafilter_1` and `chart_1`) included in the container widget (`container_1`).
	
	<pre>"container_1": {
	    "type": "container",
	    "position":{
	            "x": 0,
	            "y": 0
	    },
	    "parameters":{
	        "containedWidgets": [
	            "meafilter_1",
	            "chart_1"
	        ]
	    }
	}</pre>

`destination`	**Type** String

Property Name	Details
	Available for This Widget • link **Exposed in the Dashboard Designer's User Interface** Yes **Description** The ID of the dashboard or lens. The default value is null.
destinationType	**Type** String **Available for This Widget** • link **Exposed in the Dashboard Designer's User Interface** Yes **Description** The destination type of a link. Possible values are: • dashboard — a saved dashboard • explore — an unsaved, active exploration session of the lens • lens — a saved lens The default value is lens.
dualAxis	**Type** Boolean **Available for These Widgets** • chart (only when visualizationType is combo) **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether to include an axis for each of the two measures (true) or not (false). The default value is true.
expanded	**Type** Boolean **Available for These Widgets** • dateselector • listselector **Exposed in the Dashboard Designer's User Interface** Yes

Property Name	Details
	Description Indicates whether items in a list are displayed (`true`) or hidden (`false`). If hidden (`false`), dashboard viewers can click the list widget to view and change list items. The default value is `true`. 📝 Note: Mobile devices display items in a list, regardless of this setting.
exploreLink	**Type** Boolean **Available for These Widgets** • `chart` • `comparetable` • `listselector` • `number` • `pillbox` • `valuestable` **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether dashboard viewers can click a link to start exploring the widget as a lens (`true`) or not (`false`). The default values are: • `chart`: `true` for charts that are associated with compact form lenses; `false` if SAQL • `comparetable`: `true` for charts that are associated with compact form lenses; `false` if SAQL 📝 Note: Mobile devices display the explore link, regardless of this setting.
fit	**Type** Boolean **Available for This Widget** • `chart` (only when `visualizationType` is `scatter`, `hdot`, or `vdot`) **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether the axis of a chart is in the center of the data (`true`) or at (0, 0) (`false`). The default value is `false`.
fontSize	**Type** Integer

Property Name	Details
	Available for These Widgets • `link` • `number` • `text` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The font size of a number or of text. The default values are: • `number: 36` • `text: 26`
`hideHeaderColumn`	**Type** Boolean **Available for These Widgets** • `chart` • `valuestable` **Exposed in the Dashboard Designer's User Interface** No. Only editable via JSON. **Description** Indicates whether the first column in a raw data table—which is simply a count of rows—is hidden (`true`) or not (`false`). The default value is `false`. 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.
`imageUrl`	**Type** ConnectUri **Available for This Widget** • `box` • `container` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The document Id of the image file that you want to apply as the background. To ensure security, the image file must be uploaded to Salesforce as a document and the **Externally Available Image** option must be selected. If the option is not selected, the document is not an image, or there is no corresponding document, then nothing is displayed. The default value is null.

Property Name	Details

	Example
	This example has a container widget (`container_1`) with a background image. The image has document Id `01599000000D8HP`.
	
	``
	"container_1": {
	    "type": "container",
	    "position": {
	        "x": 0,
	        "y": 0
	    },
	    "parameters": {
	        "containedWidgets": [],
	        "imageUrl": "01599000000D8HP"
	    }
	}
	``

`includeState`	**Type** Boolean **Available for This Widget** • `link` **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether selections applied by a dashboard viewer are preserved in the `destination` after the viewer clicks the link (`true`) or not (`false`). If a selection is incompatible with the `destination` or is null, then it isn't preserved. The default value is `false`.
`instant`	**Type** Boolean **Available for These Widgets** • `dateselector` • `listselector` • `rangeselector` **Exposed in the Dashboard Designer's User Interface** Yes **Description** When a dashboard viewer interacts with a widget, indicates whether other faceted widgets immediately update (`true`) or not (`false`). When `false`, dashboard viewers must click **Update** for their changes to cascade to faceted widgets. When `true`, the **Update** button is hidden. The default values are: • `dateselector: false`

Property Name	Details
	• `listselector: true` • `rangeselector: false`
legend	**Type** Boolean **Available for This Widget** • `chart` (only when `visualizationType` is `hbar`, `vbar`, `stackhbar`, `stackvbar`, `pie`, `scatter`, `time`, `hdot`, `vdot`, `matrix`, `calheatmap`, `heatmap`, `parallelcoords`, or `stackwaterfall`) **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether to display a legend (`true`), or not (`false`). The default value is `false` for all chart types except `pivottable`. 📝 Note: Mobile devices can only display legends for `pie` widgets.
legendHideHeader	**Type** Boolean **Available for This Widget** • `chart` (only when `visualizationType` is `hbar`, `vbar`, `stackhbar`, `stackvbar`, `pie`, `scatter`, `time`, `hdot`, `vdot`, `matrix`, `calheatmap`, `heatmap`, or `parallelcoords`) **Exposed in the Dashboard Designer's User Interface** No. Only editable via JSON. **Description** Indicates whether the legend has a title (`true`) or not (`false`). The title is always the name of the dimension that the legend describes. The default value is `false` for all chart types except `pivottable`. 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.
legendWidth	**Type** Integer **Available for This Widget** • `chart` (only when `visualizationType` is `hbar`, `vbar`, `stackhbar`, `stackvbar`, `pie`, `scatter`, `time`, `hdot`, `vdot`, `matrix`, `calheatmap`, `heatmap`, or `parallelcoords`) **Exposed in the Dashboard Designer's User Interface** No. Only editable via JSON.

Property Name	Details
	Description The width of the legend area in pixels. The default value is `145` for all chart types except `pivottable`. 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.
`maxColumnWidth`	**Type** Integer **Available for These Widgets** • `chart` (only when `visualizationType` is `comparisontable`, `pivottable`, `stackhbar`, `stackvbar`, `hbar`, or `vbar`) • `comparisontable` • `valuestable` **Exposed in the Dashboard Designer's User Interface** No. Only editable via JSON. **Description** The maximum display size (in pixels) of a dimension field on a web browser of a desktop or laptop. The default value is `200`, minimum value is `20`, and maximum value is `200`. 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.
`measureField`	**Type** String **Available for These Widgets** • `listselector` • `number` • `pillbox` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The mathematical function performed on data. Specify the `measureField` in this format: `<formula>_<field>`. `<formula>` must match one of the formulas specified in the `measures step` property. Possible values for `<formula>` are: • `avg` — calculate the mathematical average (mean) • `max` — the maximum value • `min` — the minimum value • `sum` — add all the values • `unique` — count the number of unique values

Property Name	Details
	<field> must match the name of the dimension that is paired with the *<formula>* specified in measures.
	For example, if the `measures` step property is:

	``
	"measures": [
	  [
	    "sum",
	    "Profit"
	  ],
	  [
	    "avg",
	    "Discount"
	  ]
	]
	``

	Then `measureField` must be `sum_Profit` or `avg_Discount`. The `measureField` can't be `avg_Profit` because `avg` and `Profit` aren't paired together in the `measures` step property.
	The default value is null.
`minColumnWidth`	**Type** Integer **Available for This Widget** • `chart` (only when `visualizationType` is `comparisontable`, `pivottable`, `stackhbar`, `stackvbar`, `hbar`, or `vbar`) • `comparisontable` • `valuestable` **Exposed in the Dashboard Designer's User Interface** No. Only editable via JSON. **Description** The minimum display size of a dimension field in pixels. The default value is `30`. ⬛ Note: This setting doesn't apply when viewing the widget on mobile devices.
`miniBars`	**Type** Integer **Available for This Widget** • `chart` (only when `visualizationType` is `stackhbar`, `stackvbar`, `hbar`, or `vbar`) **Exposed in the Dashboard Designer's User Interface** Yes **Description** The display size in pixels of bars in bar charts.

Property Name	Details
	The default value is `0` (available only for bar charts and column charts).
`multiMetrics`	**Type** Boolean **Available for This Widget** • `chart` (only when `visualizationType` is `hbar` or `vbar`) **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether two or more measures are displayed as adjacent bars under each grouping (`true`) or as individual, adjacent graphs (`false`). The default value is `false` (available only for bar charts and column charts).
`negativeColor`	**Type** String **Available for These Widgets** • `chart` (only when `visualizationType` is `waterfall`) **Exposed in the Dashboard Designer's User Interface** Yes **Description** The color of the measure columns that have decreased in value in the chart. Specify the color in this format: `rgb(`*a*`, `*b*`, `*c*`, `*d*`)`. Using a number between zero and 255, *a* indicates how much red is in the color, *b* how much green, and *c* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, *d* indicates the level of transparency. A value of `0` is invisible and `1` is opaque. For example, `rgb(0, 0, 0, 0.93)` sets the color to a nearly opaque black. `rgb(255, 0, 0, 0.14)` sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. `#000000` indicates black in hexadecimal. `#ff0000` indicates red.
`normalize`	**Type** Boolean **Available for This Widget** • `chart` (only when `visualizationType` is `stackhbar` or `stackvbar`) **Exposed in the Dashboard Designer's User Interface** Yes

Property Name	Details
	Description Indicates whether charts are displayed using a logarithmic scale (`true`) or a linear scale (false). The default value is `false` (available only for `stackhbar` and `stackvbar`).
numberColor	**Type** String **Available for This Widget** • `number` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The font color of the number in the flex dashboard designer only. Specify the color in this format: `rgb(`***a, b, c, d***`)`. Using a number between zero and 255, ***a*** indicates how much red is in the color, ***b*** how much green, and ***c*** how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, ***d*** indicates the level of transparency. A value of `0` is invisible and `1` is opaque. For example, `rgb(0, 0, 0, 0.93)` sets the color to a nearly opaque black. `rgb(255, 0, 0, 0.14)` sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. `#000000` indicates black in hexadecimal. `#ff0000` indicates red. The default value is `#000`.
numberSize	**Type** Integer **Available for This Widget** • `number` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The font size of the number in the flex dashboard designer only. The default value is 26.
positiveColor	**Type** String **Available for These Widgets** • `chart` (only when `visualizationType` is `waterfall`)

Property Name	Details
	Exposed in the Dashboard Designer's User Interface Yes **Description** The color of the measure columns that have increased in value or remained the same in the chart. Specify the color in this format: `rgb(`*`a, b, c, d`*`)`. Using a number between zero and 255, *`a`* indicates how much red is in the color, *`b`* how much green, and *`c`* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, *`d`* indicates the level of transparency. A value of `0` is invisible and `1` is opaque. For example, `rgb(0, 0, 0, 0.93)` sets the color to a nearly opaque black. `rgb(255, 0, 0, 0.14)` sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. `#000000` indicates black in hexadecimal. `#ff0000` indicates red.
`showValues`	**Type** Boolean **Available for This Widget** • `chart` (only when `visualizationType` is `stackwaterfall` or `waterfall`) **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether to display the totals for each measure column (`true`) or not (`false`). The default value is `true`.
`splitAxis`	**Type** Boolean **Available for This Widget** • `chart` **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether each dimension in a chart is measured on its own axis (`true`) or a shared axis (`false`). Only applicable when `multiMetrics` is `true`. The default value is `false` (available only for bar charts and column charts). 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.

Property Name	Details
sqrt	**Type** Boolean **Available for This Widget** • chart (only when visualizationType is parallelcoords, hdot, vdot, time, scatter, stackhbar, stackvbar, hbar, or vbar) **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether charts are displayed using a logarithmic scale (true) or a linear scale (false). The default value is false (available only for bar charts, column charts, line charts, and time series). 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.
startColor	**Type** String **Available for These Widgets** • chart (only when visualizationType is waterfall) **Exposed in the Dashboard Designer's User Interface** Yes **Description** The color of the first measure column in the chart. Specify the color in this format: rgb(*a*, *b*, *c*, *d*). Using a number between zero and 255, *a* indicates how much red is in the color, *b* how much green, and *c* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, *d* indicates the level of transparency. A value of 0 is invisible and 1 is opaque. For example, rgb(0, 0, 0, 0.93) sets the color to a nearly opaque black. rgb(255, 0, 0, 0.14) sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. #000000 indicates black in hexadecimal. #ff0000 indicates red.
step	**Type** String **Available for These Widgets** • chart • comparetable • dateselector • globalfilters

Property Name	Details
	• `listselector` • `number` • `pillbox` • `rangeselector` • `valuestable` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The name of the lens that supplies data for the widget. The default value is null.
`stretch`	**Type** Boolean **Available for This Widget** • `box` **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether an image's width and height are set to the same values of the widget's width and height (`true`) or not (`false`). The default value is `false`.
`stretchImage`	**Type** Boolean **Available for This Widget** • `container` **Exposed in the Dashboard Designer's User Interface** Yes **Description** Indicates whether an image's width and height are set to the same values of the widget's width and height (`true`) or not (`false`). The default value is `false`.
`text`	**Type** String **Available for This Widget** • `link` • `text` **Exposed in the Dashboard Designer's User Interface** Yes

Property Name	Details
	Description The message rendered in a text widget. For example, if `text` is assigned the value "`Hello, World!`", then "Hello, World!" appears in the text widget. The default value is null.
`textAlignment`	**Type** String **Available for This Widget** • `number` • `text` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The alignment of text. Possible values include `left`, `center`, and `right`. If no value is specified, text alignment defaults to center. The default values are: • `number: right` • `text: center`
`textColor`	**Type** String **Available for These Widgets** • `link` • `number` • `text` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The font color of text. Specify the color in this format: `rgb(`*a*`,` *b*`,` *c*`,` *d*`)`. Using a number between zero and 255, *a* indicates how much red is in the color, *b* how much green, and *c* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, *d* indicates the level of transparency. A value of `0` is invisible and `1` is opaque. For example, `rgb(0, 0, 0, 0.93)` sets the color to a nearly opaque black. `rgb(255, 0, 0, 0.14)` sets the color to a nearly invisible red.

Property Name	Details
	Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. `#000000` indicates black in hexadecimal. `#ff0000` indicates red. The default value is `#000`.
`title`	**Type** String **Available for These Widgets** • `dateselector` • `listselector` • `number` • `pillbox` • `rangeselector` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The title of a widget. The default value is null.
`titleColor`	**Type** String **Available for This Widget** • `number` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The font color of the title in the flex dashboard designer only. Specify the color in this format: `rgb(`*a*`, `*b*`, `*c*`, `*d*`)`. Using a number between zero and 255, *a* indicates how much red is in the color, *b* how much green, and *c* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, *d* indicates the level of transparency. A value of `0` is invisible and `1` is opaque. For example, `rgb(0, 0, 0, 0.93)` sets the color to a nearly opaque black. `rgb(255, 0, 0, 0.14)` sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. `#000000` indicates black in hexadecimal. `#ff0000` indicates red. The default value is `#000`.

Property Name	Details
titleSize	**Type** Integer **Available for This Widget** • number **Exposed in the Dashboard Designer's User Interface** Yes **Description** The font size of the title in the flex dashboard designer only. The default value is 26.
totalColor	**Type** String **Available for These Widgets** • chart (only when visualizationType is waterfall) **Exposed in the Dashboard Designer's User Interface** Yes **Description** The color of the total measure column in the chart. Specify the color in this format: rgb(*a*, *b*, *c*, *d*). Using a number between zero and 255, *a* indicates how much red is in the color, *b* how much green, and *c* how much blue. A value of 0 indicates the absence of a color, and a value of 255 indicates the full expression of a color. Using a number between zero and one, *d* indicates the level of transparency. A value of 0 is invisible and 1 is opaque. For example, rgb(0, 0, 0, 0.93) sets the color to a nearly opaque black. rgb(255, 0, 0, 0.14) sets the color to a nearly invisible red. Alternatively, the color can be set using hexadecimal notation. When using hexadecimal notation, transparency can't be set. All hexadecimal colors default to opaque. #000000 indicates black in hexadecimal. #ff0000 indicates red.
totals	**Type** Boolean **Available for These Widgets** • chart (only when visualizationType is pivottable) • comparetable • valuestable **Exposed in the Dashboard Designer's User Interface** Yes

Property Name	Details
	Description Indicates whether to include a row that displays the sum of all the values in each measure column (`true`) or not (`false`). The default value for `chart` is `false` (available only for `pivottable`). 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.
`trellis`	**Type** Boolean **Available for This Widget** • `chart` **Exposed in the Dashboard Designer's User Interface** Yes **Description** When a lens has two or more groupings and one measure, indicates whether the last grouping displays on its own axis (`true`) or on the same axis as other groupings (`false`). The default value for `chart` is `false` (available only for bar charts and column charts). 📝 Note: This setting doesn't apply when viewing the widget on mobile devices.
`videoSize`	**Type** String **Available for This Widget** • `url` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The dimensions of a YouTube video. Possible values are: • `(4/3) 240 x 180` • `(4/3) 420 x 315` • `(4/3) 480 x 360` • `(4/3) 640 x 480` • `(4/3) 960 x 720` • `(16/9) 320 x 180` • `(16/9) 560 x 315` • `(16/9) 640 x 360` • `(16/9) 853 x 480` • `(16/9) 1280 x 720` The default value is `(4/3) 240 x 180`.

Property Name	Details
	📝 **Note:** Mobile devices don't display `url` widgets.
`visualizationType`	**Type** ConnectWaveChartTypeEnum **Available for These Widgets** • `chart` • `link` **Exposed in the Dashboard Designer's User Interface** Yes **Description** The type of chart used to show data. Possible values are: • `calheatmap*` — calendar heat map • `combo` — lines and bars to show multiple metrics • `comparisontable` — comparison table • `hbar` — horizontal bar • `hdot*` — horizontal dot plot • `heatmap*` — heat map • `matrix*` — matrix • `parallelcoords*` — parallel coordinates • `pie` — donut • `pivottable*` — pivot table • `scatter` — scatter plot • `stackhbar` — stacked horizontal bar • `stackvbar` — stacked vertical bar • `stackwaterfall` — stacked waterfall • `time` — timeline • `valuestable` — raw data table in original dashboard designer (values table in flex dashboard designer) • `vbar` — vertical bar • `vdot*` — vertical dot plot • `waterfall` — waterfall 📝 **Note:** The flex dashboard designer doesn't support chart types with an asterisk (*). If you specify an unsupported type, the designer replaces it with `hbar` in the dashboard.
`url`	**Type** ConnectUri **Available for This Widget** • `url`

Property Name	Details
	Exposed in the Dashboard Designer's User Interface Yes **Description** The URL of a YouTube video. The default value is null. 📝 Note: Mobile devices don't display `url` widgets.

SEE ALSO:

[Widgets](#)

QUERY

The `query` section defines the query for that step.

The properties of the `query` section of a dashboard JSON file are:

Field Name	Description
aggregateFilters	Automatically generated. Don't modify.
autoFilter	Enables filters from compact-form query lenses and Scope widgets to be applied to the faceted SAQL query lens. To apply filters from compact-form query lenses to the SAQL query lens, set `autoFilter` and `isFacet` to `true`. To apply filters from Scope widgets to the SAQL query lens, set `autoFilter` and `useGlobal` to `true`. If `autoFilter` is set to `false` or not specified, filters from compact-form query lenses and Scope widgets are not applied to the SAQL query lens.
dimensions	The dimensions to use are specified this way: `"dimensions": [ "Department" ]`
facet_filters	Automatically generated. Don't modify.
filters	The filter conditions to apply to the data. Here's an example of a simple filter condition to include only rows that have the destination `"SFO"`, `"LAX"`, `"ORD"`, or `"DFW"`: `"filters": [["dest", ["SFO", "LAX", "ORD", "DFW"]]]`
formula	Formula is used with the `multi` step type in a step for a compare table. A `multi` type step includes multiple subqueries. You can use the basic mathematical operators `*`, `/`, `-`, `+`, `(`, and `)` to create a formula to reference other subqueries in the step. To reference other subqueries, use the automatically assigned names: "A" is the first query, "B" is the second query, and so on.

```
 "step_comptable": {
       "type": "multi",
       "datasets":[{"name":"opp"}],
       "isFacet": true,
       "useGlobal": true,
       "query": {
         "columns": [
            {
               "header": "Opptys Won",
               "query": {
                 "pigql": null,
                 "filters": [["StageName", ["5 - Closed-Won"]], ["Close
Date", [[["year", -1], ["year", 0]]]]],
                 "measures": [["count", "*"]],
                 "values": [],
                 "groups": ["Owner-Name"],
                 "formula": null,
                 "order": []
```

Field Name	Description

```
                  }
                }, {
                  "header": "Opptys Won ($)",
                  "query": {
                    "pigql": null,
                    "filters": [["StageName", ["5 - Closed-Won"]]],
                    "measures": [["sum", "Amount"]],
                    "values": [],
                    "groups": ["Owner-Name"],
                    "formula": null,
                    "order": []
                  }
                }, {
                  "sort": {
                    "asc": false,
                    "inner": false
                  },
                  "header": "Opptys Won ($)",
                  "showBars": true,
                  "query": {
                    "pigql": null,
                    "filters": [["StageName", ["5 - Closed-Won"]]],
                    "measures": [["sum", "Amount"]],
                    "values": [],
                    "groups": ["Owner-Name"],
                    "formula": null,
                    "order": []
                  }
                }, {
                  "header": "Opptys Lost ($)",
                  "query": {
                    "pigql": null,
                    "filters": [["StageName", ["5 - Closed-Lost"]]],
                    "measures": [["sum", "Amount"]],
                    "values": [],
                    "groups": ["Owner-Name"],
                    "formula": null,
                    "order": []
                  }
                }, {
                  "header": "Opptys Lost ($)",
                  "showBars": true,
                  "query": {
                    "pigql": null,
                    "filters": [["StageName", ["5 - Closed-Lost"]]],
                    "measures": [["sum", "Amount"]],
                    "values": [],
                    "groups": ["Owner-Name"],
                    "formula": null,
                    "order": []
                  }
                }, {
```

Field Name	Description

	``
	          "header": "Win-Loss (%)",
	          "query": {
	            "groups": ["Owner-Name"],
	            "filters": [["StageName", ["5 - Closed-Lost"]]],
	            "measures": [["sum", "Amount"]],
	            "values": [],
	            "pigql": null,
	            "formula": "B/(B+D)*100",
	            "order": []
	          }
	        }
	      ]
	    }
	  }
	},
	``

groups	The dimension to group by. For example, `"groups": ["carrier"]`. Specify groups for both compact and SAQL query formats.
limit	The number of results to return. For example, `"limit": 10`. The results that the limit statement returns aren't automatically ordered, so use this statement only with data that has been ordered.

measures	The measures to use are specified this way:
	
	``
	"count", "*", null, {
	    "display": "% of total flights"
	}
	``
	
	Specify for both compact and SAQL query formats. Specify for SAQL queries so that the associated chart widget can render the correct projections. You can change the UI label of a measure by setting the `display` option.

order	The sort order is specified this way: ``` "order": [[-1, { "ascending": false }]] ``` The value –1 indicates that the ordering is done for the first measure. To order the results in ascending order, set `ascending` to `true`. To order the results in descending order, set `ascending` to `false`. If you don't want to impose a specific order, specify empty brackets this way: `"order": []`. Can be specified for both compact and SAQL query formats.
pigql	The SAQL query to use. The SAQL language is a real-time query language that uses data flow to align results. It enables on-demand analysis of data that's stored in datasets. 📝 Note: You can enable faceting on a lens created from a SAQL query. However, if the SAQL query is based on multiple datasets, only the first dataset specified in the `datasets` field is faceted.

values	Values are used with the `grain` step type in a step for a values table widget. Values list the columns to include in a grain or values table. For example:
	
	``
	"step_grain": {
	  "type": "grain",
	  "datasets":[{"name":"opp"}],
	``

34

Field Name	Description

```
  "query": {
    "values": ["Amount", "Owner-Name", "Name", "Account-Name",
"StageName", "ForecastCategory", "Current Age", "Time to Win"],
    }
}
```

Specify values for both compact and SAQL query formats.

Within the `query` section of a step, you can manually insert bindings. To do so, use templates—expressions that are embedded in double braces ({{ }}) and that get replaced with the current state of the step that they're attached to. Here's an example:

```
"filters": [["carrier", "{{ selection(step1) }}"], ["dest", "{{ filter(step1, 'dest') }}"],
 ["origin", "{{ filter(step1, 'origin') }}"]]
```

Query Example
This example shows a dashboard that contains two queries.

Compare Table Example (Mobile)
This example shows a snippet with a single, unified SAQL query for creating a Compare Table on a mobile client.

Query Example

This example shows a dashboard that contains two queries.

The first bar chart is connected to a step (`"step3"`) that contains a query that uses SAQL. The second bar chart is connected to a step (`"step2"`) that contains a compact form query. Both the compact and the SAQL steps have selection filters that are bound to `"step1"`. Clicking one chart filters the others.

In `"step3"`, the full SAQL query is placed within the `"piqql":` reference. The SAQL query is used instead of the compact query references. However, the compact form elements of `"groups"` and `"measures"` still need to be specified, so that the associated chart widget can render the correct projections. (For a `"grain"` type query, `"values"` is always specified.) In this example, the `'sum_miles'` and `'count'` projections in the SAQL query are then referenced in measures as `[["sum", "miles"]], ["count", "*"]]`. Measure projections in the SAQL always include the aggregation underscore (_) and the name of the measure (`'sum_miles'`) so that they can be referenced in the compact form `"measures": [["sum", "miles"]`.

For more information about SAQL, see the *SAQL Reference*.
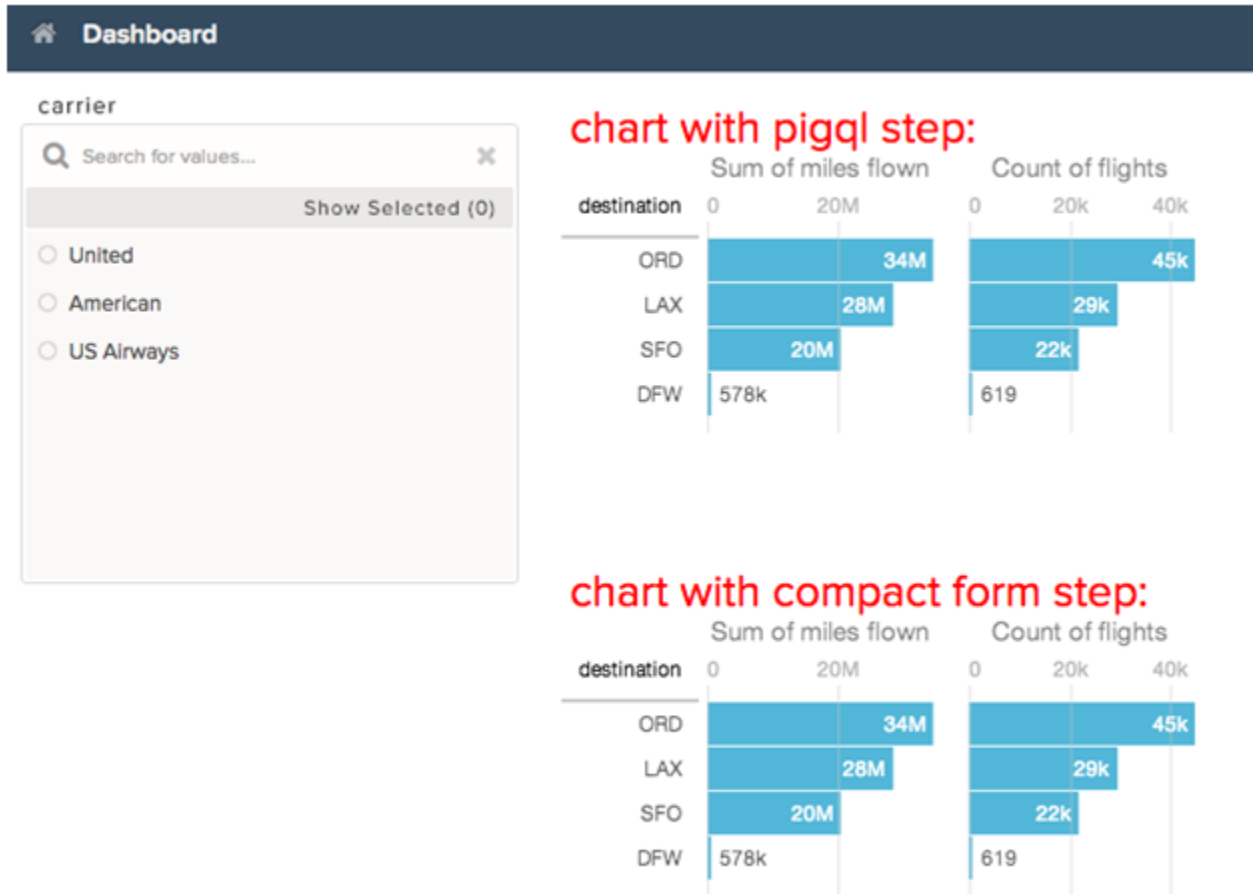
```
{
  "steps": {
    "step1": {
      "type": "aggregate",
      "datasets":[{"name":"airline"}],
      "query": {
        "groups": ["carrier"],
        "filters": [["dest", ["SFO", "LAX", "ORD", "DFW"]]],
        "measures": [["count", "*"]],
        "order": [
          [
            -1, {
              "ascending": false
            }
          ]
        ],
        "limit": 3
      }
    },
    "step2": {
      "type": "aggregate",
      "datasets":[{"name":"airline"}],
```

```
      "query": {
        "groups": ["dest"],
        "filters": [["carrier", "{{ selection(step1) }}"], ["dest", "{{ filter(step1,
'dest') }}"], ["origin", "{{ filter(step1, 'origin') }}"]],
        "measures": [["sum", "miles"], ["count", "*"]],
        "order": [
          [
            -1, {
              "ascending": false
            }
          ]
        ]
      }
    },
    "step3": {
      "type": "aggregate",
      "datasets":[{"name":"airline"}],
      "query": {
        "pigql": "q = load \"airline\";\nq = filter q by 'carrier' in {{ selection(step1)
}};\nq = filter q by 'dest' in {{ filter(step1, 'dest') }};\nq = filter q by 'origin' in
{{ filter(step1, 'origin') }};\nq = group q by 'dest';\nq = foreach q generate 'dest' as
'dest', sum('miles') as 'sum_miles', count() as 'count';\nq = order q by 'count' desc;",

        "groups": ["dest"],
        "measures": [["sum", "miles"], ["count", "*"]]
      }
    }
  },
  "widgets": {
    "barchart1": {
      "type": "listselector",
      "position": {
        "x": 10,
        "y": 10,
        "w": "270",
        "h": "180"
      },
      "parameters": {
        "step": "step1"
      }
    },
    "text2": {
      "type": "text",
      "position": {
        "x": 310,
        "y": 10
      },
      "parameters": {
        "text": "chart with pigql step:",
        "textColor": "#f00"
      }
    },
    "barchart2": {
      "type": "chart",
```

```
          "position": {
            "x": 310,
            "y": 30,
            "w": "400",
            "h": "280"
          },
          "parameters": {
            "step": "step2",
            "visualizationType": "hbar"
          }
        },
        "text3": {
          "type": "text",
          "position": {
            "x": 310,
            "y": 280
          },
          "parameters": {
            "text": "chart with compact form step:",
            "textColor": "#f00"
          }
        },
        "barchart3": {
          "type": "chart",
          "position": {
            "x": 310,
            "y": 300,
            "w": "400",
            "h": "280"
          },
          "parameters": {
            "step": "step3",
            "visualizationType": "hbar"
          }
        }
      }
    }
}
```

Compare Table Example (Mobile)

This example shows a snippet with a single, unified SAQL query for creating a Compare Table on a mobile client.

This example uses a pigql definition under globalQuery field to illustrate a unified SAQL query for creating a simple, two-column Compare Table.

Industry	Sum of LeadScore ▾	Avg of LeadScore ⌄
High Tech		3.0679
Fin Svcs		3.1796
Mfg		2.8361
Healthcare		3.5238
Prof Svcs		3.4258
Consumer		2.3604

```
"compare_2": {
    "isFacet": true,
    "isGlobal": false,
    "selectMode": "single",
    "type": "multi",
    "useGlobal": true,
    "start": null,
    "datasets": [
        {
            "name": "Honeywell_Recent_Deals1"
        }
    ],
    "visualizationParameters": {
        "visualizationType": "comparisontable"
    },
    "columns": [
        {
            "header": "Sum of LeadScore",
            "query": {
                "measures": [
                    [
                        "max",
                        "LeadScore"
                    ]
                ],
                "groups": [
                    "Industry"
                ]
            },
            "showBars": true
        },
        {
            "header": "Avg of LeadScore",
            "query": {
                "measures": [
                    [
                        "avg",
                        "LeadScore"
                    ]
```

```
                    ],
                    "groups": [
                        "Industry"
                    ]
                },
                "showBars": false
            }
        ],
        "globalQuery": {
            "pigql": "q = load \"Honeywell_Recent_Deals1\"; q = group q by 'Industry'; q =
filter q by 'Industry' in [\"Consumer\",\"Fin Svcs\",\"Mfg\",\"High
Tech\",\"Healthcare\",\"Prof Svcs\"];  q = foreach q generate 'Industry' as 'Industry',
avg('LeadScore') as 'avg_LeadScore', sum('LeadScore') as 'sum_LeadScore'; q = limit q
2000;"
        }
},
```

The Compare Table has the following limitations:

- Only these functions can be included: +, -, *, /, ( ).

- On mobile devices, do not use SAQL at the column level. A global SAQL definition is supported, or use the compact form in each column.

- On mobile devices, the Compare Table is read-only.

For more information about SAQL, see the *SAQL Reference*.

BINDINGS

After you define steps, you bind them to the widgets.

The kinds of bindings are:

- Selection binding
- Results binding

Selection Binding

When a user makes a selection on a widget in a dashboard, that selection value can be used to update other steps and widgets to make the dashboard interactive. This action is referred to as faceting.

When you build a dashboard with the dashboard builder UI, by default, everything is faceted. The "isFaceted" option for each step takes care of bidirectional selection bindings between steps of the same dataset. However, you can modify a dashboard JSON file directly to manually specify the relationships between the various steps to achieve the following.

- Selection bindings between steps of different datasets
- Unidirectional selection binding
- Selection binding for a static step

📝 Note: You can't configure selection binding on a multi-metric widget. If you do, an error occurs.

Results Binding

Results binding is used to filter a step by using the values that result from another step. This type of binding is typically used across multiple datasets. An example of when results binding is useful is when you want to filter opportunities by top-selling products.

```
"step_all_salesreps": {
    "type": "aggregate",
    "datasets": [{"name":"opp"}],
    "query": {
        "groups": ["Owner-Name"],
        "filters": [
            ["StageName", ["5 - Closed-Won"]],
            ["Products", "{{results(step_top5_products) }}"]
        ],
        "measures": [["sum", "Amount"]]
    }
}
```

In the following example, the resulting sum of miles from the first step (`"all_miles"`) is used in the second step to calculate the average.

```
"steps": {
    "all_miles": {
        "type": "aggregate",
        "datasets":[{"name":"airline"}],
```

```
    "query": {
      "measures": [["sum", "miles"], ["count", "*"]]
    }
  },
  "step_percent": {
    "type": "aggregate",
    "datasets":[{"name":"airline"}],
    "query": {
      "pigql": "q = load \"airline\";\nq = group q by 'carrier';\nq =
          foreach q generate 'carrier' as 'carrier', sum('miles')/{{
          value(results(all_miles, 'sum_miles')) }} * 100 as 'sum_miles',
          count()/{{ value(results(all_miles, 'count')) }} * 100 as 'count';\nq =
          order q by 'sum_miles' desc;",
      "groups": ["carrier"],
      "order": [
        [
          ["sum", "miles"], {
            "ascending": false
          }
        ]
      ],
      "measures": [
        [
          "sum", "miles", null, {
            "display": "% of total miles"
          }
        ], [
          "count", "*", null, {
            "display": "% of total flights"
          }
        ]
      ]
    }
  }
}
```

Selection Binding in a Static Step

Almost all parts of a step can include a selection binding to the results of a prior query.

In an aggregate query, the fields that can be included in a selection binding are:

- Group

- Measure
- Filters
- Sort
- Limit

Use Static Steps for Binding Any Part of a Query

This example shows a dashboard with static steps and selection bindings in multiple parts of a query.
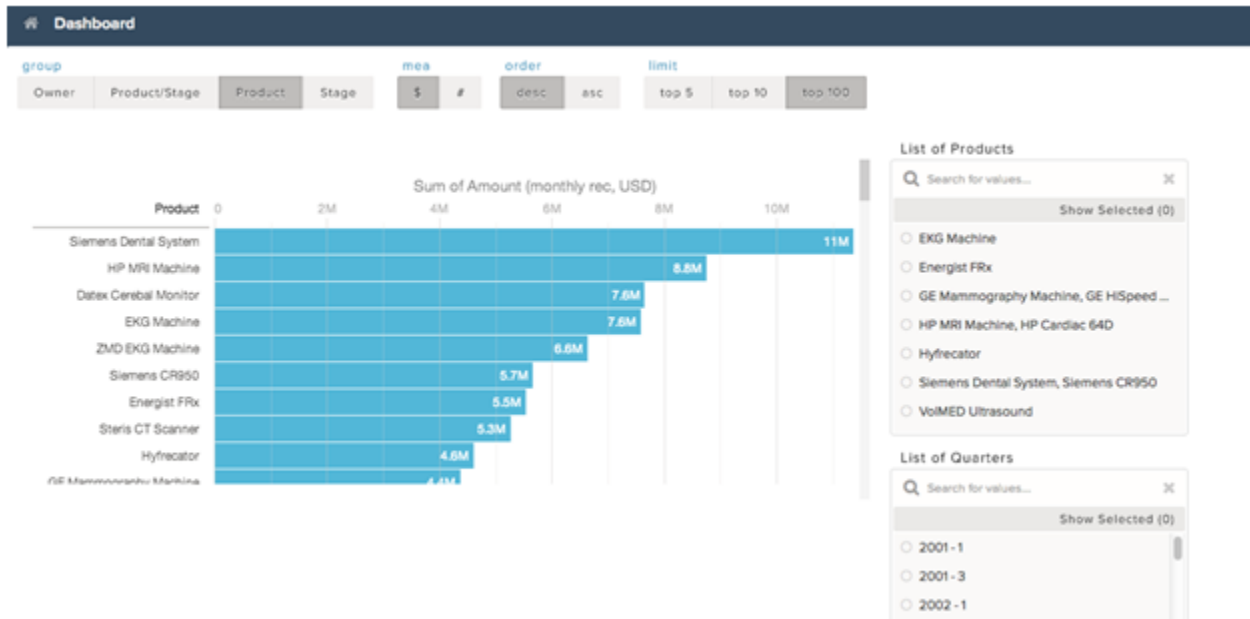
In the following example:

- The static step `step_filter_dim` populates the `"List of Products"` list selector. It includes options that have multiple values.
- The static step `step_group` populates the group toggle selector. `"Product"` is the default value when the dashboard is initialized, because the `start` value is `"Product"`. The `display` values change the display name in the user interface.
- The static step `step_measure` populates the measure toggle selector.
- The static step `step_order` populates the order toggle selector.
- The static step `step_limit` populates the limit toggle selector.
- The aggregate step query `step_quarterly_bookings` is grouped by close-date year and quarter.
- The aggregate step query `step_top_10` has groupings that depend on the selection option from the static `step_group`. The `start` value is the `"Product"` grouping (based on `step_group`).

```
{
  "steps": {
    "step_filter_dim": {
      "type": "static",
      "dimensions": [ "Product" ],
      "datasets":[{"name":"opp"}],
```

```
    "selectMode": "single",
    "values": [
      {
        "value": ["EKG Machine"]
      }, {
        "value": ["Energist FRx"]
      }, {
        "value": ["GE Mammography Machine", "GE HiSpeed DXi", "GE Stress System"]
      }, {
        "value": ["HP MRI Machine", "HP Cardiac 64D"]
      }, {
        "value": ["Hyfrecator"]
      }, {
        "value": ["Siemens Dental System", "Siemens CR950"]
      }, {
        "value": ["VolMED Ultrasound"]
      }
    ],
    "isFacet": true
  },
  "step_group": {
    "type": "static",
    "values": [
      {
        "display": "Owner",
        "value": ["Owner-Name"]
      }, {
        "display": "Product/Stage",
        "value": ["Product", "StageName"]
      }, {
        "display": "Product",
        "value": ["Product"]
      }, {
        "display": "Stage",
        "value": ["StageName"]
      }
    ],
    "start": [["Product"]],
    "selectMode": "single"
  },
  "step_measure": {
    "type": "static",
    "values": [
      {
        "display": "$",
        "value": [["sum", "Amount"]]
      }, {
        "display": "#",
        "value": [["count", "*"]]
      }
    ],
    "start": [[["sum", "Amount"]]],
    "selectMode": "singlerequired"
  },
```

```
"step_order": {
  "type": "static",
  "values": [
    {
      "display": "desc",
      "value": false
    }, {
      "display": "asc",
      "value": true
    }
  ],
  "selectMode": "singlerequired"
},
"step_limit": {
  "type": "static",
  "values": [
    {
      "display": "top 5",
      "value": 5
    }, {
      "display": "top 10",
      "value": 10
    }, {
      "display": "top 100",
      "value": 100
    }
  ],
  "start": [100],
  "selectMode": "singlerequired"
},
"step_quarterly_bookings": {
  "type": "aggregate",
  "datasets":[{"name":"opp"}],
  "query": {
    "groups": [["CloseDate_Year", "CloseDate_Quarter"]],
    "measures": [["sum", "Amount"]]
  },
  "isFacet": true,
  "useGlobal": true
},
"step_top_10": {
  "type": "aggregate",
  "datasets":[{"name":"opp"}],
  "query": {
    "groups": "{{ selection(step_group) }}",
    "measures": "{{ selection(step_measure) }}",
    "order": [
      [
        -1, {
          "ascending": "{{ value(selection(step_order)) }}"
        }
      ]
    ],
    "limit": "{{ value(selection(step_limit)) }}"
```

```
          },
          "isFacet": true
        }
      },
      "widgets": {
        "sel_list_filter_dim": {
          "type": "listselector",
          "position": {
            "x": 860,
            "y": 90,
            "w": "290",
            "h": "288"
          },
          "parameters": {
            "step": "step_filter_dim",
            "title": "List of Products",
            "expanded": true,
            "instant": true
          }
        },
        "sel_list_filter_compound_dim": {
          "type": "listselector",
          "position": {
            "x": 860,
            "y": 390,
            "w": "290",
            "h": "288"
          },
          "parameters": {
            "step": "step_quarterly_bookings",
            "title": "List of Quarters",
            "expanded": true,
            "instant": true
          }
        },
        "sel_group": {
          "type": "pillbox",
          "position": {
            "x": 10,
            "y": 10
          },
          "parameters": {
            "title": "group",
            "step": "step_group"
          }
        },
        "sel_measure": {
          "type": "pillbox",
          "position": {
            "x": 380,
            "y": 10
          },
          "parameters": {
            "title": "mea",
```

46

```
          "step": "step_measure"
        }
      },
      "sel_order": {
        "type": "pillbox",
        "position": {
          "x": 480,
          "y": 10
        },
        "parameters": {
          "title": "order",
          "step": "step_order",
          "start": true
        }
      },
      "sel_limit": {
        "type": "pillbox",
        "position": {
          "x": 620,
          "y": 10
        },
        "parameters": {
          "title": "limit",
          "step": "step_limit"
        }
      },
      "widget1": {
        "type": "chart",
        "position": {
          "x": 10,
          "y": 110,
          "w": "830",
          "h": "330"
        },
        "parameters": {
          "visualizationType": "hbar",
          "step": "step_top_10"
        }
      }
    }
  }
}
```

Bind a Static Filter and Group Selector to a Query

Static filters or group selectors can be bound to a query that's written in SAQL.

Templates are expressions, embedded in double braces ({{ }}), that get replaced with the current state of the step that they're attached to.
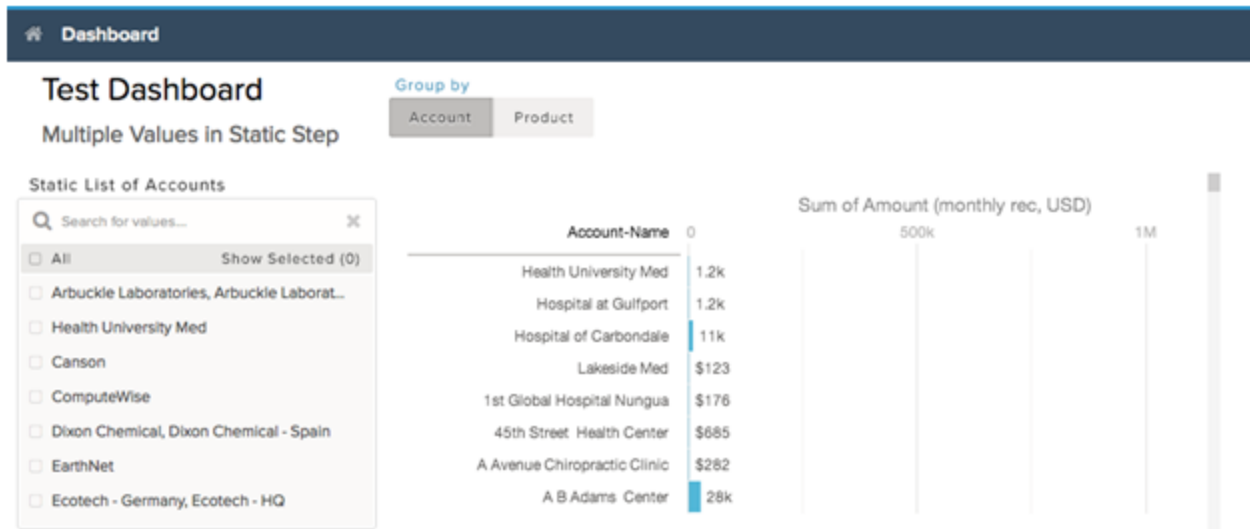
For example, this dashboard contains a static filter widget that contains a list of accounts. The dashboard also contains a group selector widget that lets users indicate whether to group by account or product. When a user makes a selection, the chart is updated accordingly. The part of the query that controls the filtering is:

```
q = filter q by 'Account-Name' in {{ selection(step_Account_Owner_Name_2) }};
```

The step that's named *step_Account_Owner_Name_2* is configured as a selection binding so that it picks up the current selection state. Because it's within the double braces, the value of that selection is substituted and used in the query.

The part of the query that controls the grouping is:

```
q = group q by {{ single_quote(value(selection(step_StageName_3))) }};
q = foreach q generate {{ single_quote(value(selection(step_StageName_3))) }} as {{
value(selection(step_StageName_3)) }}, sum('Amount') as 'sum_Amount', count() as 'count'";
```

If a user selects Product in the group selector widget, the actual query that's passed to the query engine contains:

```
q = group q by 'Product';
q = foreach q generate 'Product' as "Product", sum('Amount') as 'sum_Amount', count() as
'count';
```

📝 Note: To view the query that's used to update the chart, open your browser's JavaScript console and type *edge.log.query=true*. On the dashboard, select a different group. The new query appears in the console unless the query is cached.

```
 "steps": {
   "step_Account_Name_1": {
     "isFacet": false,
     "query": {
       "pigql": "q = load \"opp\";\nq = filter q by 'Account-Name' in {{
selection(step_Account_Owner_Name_2) }};\nq = group q by {{
single_quote(value(selection(step_StageName_3))) }};\nq = foreach q generate {{
single_quote(value(selection(step_StageName_3))) }} as {{ value(selection(step_StageName_3))
 }}, sum('Amount') as 'sum_Amount', count() as 'count'",
       "groups": "{{ selection(step_StageName_3) }}",
       "measures": [["sum", "Amount"]]
```

```
      },
      "visualizationParameters": {
        "visualizationType": "hbar"
      },
      "selectMode": "none",
      "useGlobal": true,
      "datasets":[{"name":"opp"}],
      "type": "aggregate",
      "isGlobal": false
    },
    "step_Account_Owner_Name_2": {
      "dimensions": [ "Account-Name" ],
      "isFacet": false,
      "values": [
        {
          "value": ["Lakeside Med", "Hospital at Gulfport", "Hospital at Carbondale"],
          "display": "Arbuckle Laboratories, Arbuckle Laboratories - Austria, Arbuckle
Laboratories - France"
        }, {
          "value": ["Health University Med"],
          "display": "Health University Med"
        }, {
          "value": ["Canson"],
          "display": "Canson"
        }, {
          "value": ["ComputeWise"],
          "display": "ComputeWise"
        }, {
          "value": ["Dixon Chemical", "Dixon Chemical - Spain"],
          "display": "Dixon Chemical, Dixon Chemical - Spain"
        }, {
          "value": ["EarthNet"],
          "display": "EarthNet"
        }, {
          "value": ["Ecotech - Germany", "Ecotech - HQ"],
          "display": "Ecotech - Germany, Ecotech - HQ"
        }
      ],
      "selectMode": "multi",
      "useGlobal": true,
      "datasets":[{"name":"opp"}],
      "type": "static",
      "isGlobal": false
    },
    "step_StageName_3": {
      "isFacet": false,
      "values": [
        {
          "value": ["Account-Name"],
          "display": "Account"
        }, {
          "value": ["Product"],
          "display": "Product"
        }
```

```
        ],
        "useGlobal": true,
        "datasets":[{"name":"opp"}],
        "type": "static",
        "selectMode": "singlerequired",
        "isGlobal": false
    }
  }
```

Binding a Date Picker and Static Dates

You can use selection bindings to filter lenses for dates from a date picker lens or a static absolute or relative date step.

These examples demonstrate how to bind a date picker lens to filter another query and a static relative date step to another query.

Binding a Date Picker to a Compact and SAQL Query

In this example, a date picker lens filters a time chart lens using a selection() binding. The lens for the date picker is:

```
"step_for_datePicker": {
      "type": "aggregate",
      "datasets":[{"name":"opp"}],
      "query": {
       "groups": [
        [
         "CloseDate_Year",
         "CloseDate_Month"
        ]
       ],
       "measures": [
        [
         "count",
         "*"
        ]
       ],
       "limit": 50
      },
      "start": [
       [
        [
         "year",
         -3
        ],
        [
         "year",
         1
        ]
       ]
      ]
     },
```

To filter another lens by the selection in the date picker, add the following code into a compact or SAQL step.

```
{{selection(step_for_datePicker)}}
```

The compact form looks like the following.

```
"step_compact_filtered_by_date_saql": {
      "type": "aggregate",
      "datasets":[{"name":"OpportunityWithAccount"}],
      "query": {
       "groups": [
        [
         "CloseDate_Year",
         "CloseDate_Month"
        ]
       ],
       "measures": [
        [
         "count",
         "*"
        ]
       ],
       "filters": [
        [
         "CloseDate",
         "{{ selection(step_for_datePicker) }}"
        ]
       ],
       "limit": 50
      }
     }
```

The SAQL looks like the following.

```
"step_date_saql_binding": {
      "type": "aggregate",
      "query": {
      "pigql": "q = load \"OpportunityWithAccount\";\nq = filter q by date('CloseDate_Year',
 'CloseDate_Month', 'CloseDate_Day') in {{selection(step_for_datePicker)}};\nq = group q
by ('CloseDate_Year', 'CloseDate_Month');\nq = foreach q generate 'CloseDate_Year' + \"~~~\"
 + 'CloseDate_Month' as 'CloseDate_Year~~~CloseDate_Month', count() as 'count';\nq = limit
 q 2000;",
      "groups": [
        [
         "CloseDate_Year",
         "CloseDate_Month"
        ]
       ],
       "measures": [
        [
         "count",
         "*"
        ]
       ]
      },
      "isFacet": false,
```

```
      "useGlobal": true
    }
  }
```

📝 **Note:** The date dimension that the selection is filtering (in this example, `"CloseDate"`) must be the same dimension name that's used in `"groups"` in the date picker lens.

Binding a Static Date List Selector to Filter Other Compact or SAQL Lenses

In this example, a selection from a list or toggle lens of predefined date ranges filters another lens in a dashboard. The following sample shows a selection() binding from a static toggle button lens (`"step_date_static_with_start"`) to a bar chart lens in compact form (`"compact_step_faceted_by_static"`) or SAQL (`"saql_step_faceted_by_static"`). Each value is a relative date range, for example, five years ago (`"year", -5`) until this year (`"year", 0`).

```
"step_date_static_with_start": {
      "type": "static",
      "values": [
       {
        "display": "-6 years",
        "value": [
         [
          [
           "year",
           -6
          ],
          [
           "year",
           0
          ]
         ]
        ]
       },
       {
        "display": "-5 years",
        "value": [
         [
          [
           "year",
           -5
          ],
          [
           "year",
           0
          ]
         ]
        ]
       },
       {
        "display": "-4 years",
        "value": [
         [
          [
           "year",
```

52

```
         -4
      ],
      [
       "year",
       0
      ]
     ]
    ]
   }
  ],
  "selectMode": "singlerequired",
  "start": [
   [
    [
     [
      "year",
      -5
     ],
     [
      "year",
      0
     ]
    ]
   ]
  ]
 }
```

You can then use the previous sample to filter another compact or SAQL step on selection by using the selection() binding.

```
{{selection(step_date_static_with_start)}}
```

The compact form looks like the following.

```
"compact_step_faceted_by_static": {
      "type": "aggregate",
      "datasets":[{"name":"opp"}],
      "query": {
       "groups": [
        "Product"
       ],
       "filters": [
        [
         "CreatedDate",
         "{{selection(step_date_static_with_start)}}"
        ]
       ],
       "measures": [
        [
         "sum",
         "Amount"
        ]
       ],
       "limit": 2000
      },
```

```
      "isFacet": false
    }
```

The SAQL selection binding is:

```
"saql_step_faceted_by_static": {
    "type": "aggregate",
    "query": {
    "pigql": "q = load \"opp\";\nq = filter q by date('CreatedDate_Year',
'CreatedDate_Month', 'CreatedDate_Day') in {{selection(step_date_static_with_start)}};\nq
 = group q by 'Product';\nq = foreach q generate 'Product' as 'Product', sum('Amount') as
 'sum_Amount', count() as 'count';\nq = limit q 2000;",
    "groups": [
     "Product"
    ],
    "measures": [
     [
      "sum",
      "Amount"
     ]
    ]
    },
    "isFacet": false,
    "useGlobal": true
   },
```

Binding Operations

You can use several more operations with results and selection bindings to extract the correct results.

value()

The `value()` operation is used to get a selector array value and convert it to a single value. If the selector array value is empty, the operation returns all values. Because the value() operation can return multiple values when the selector array value is empty, use `in`, not ==, like in this example:

```
q = filter q by 'Owner Name' in {{ value(selection(step_StageName_3))}}
```

single_quote()

The `single_quote()` operation is typically used in selection bindings in a SAQL step to correctly format the "`group`" and "`foreach generate`" lines in the query. The `single_quote()` operation takes an array of values and converts double quotes into single quotes and square brackets into parentheses. For example: `"Owner-Name"` converts to `'Owner-Name'`, and `["Owner-Name", "Owner-Region"]` converts to `('Owner-Name', 'Owner-Region')`.

Consider the following static selector, with the array values `["Account-Name"]` and `["Product"]`:

```
{
   "step_StageName_3": {
       "isFacet": false,
       "values": [
           {
```

```
                "value": [
                    "Account-Name"
                ],
                "display": "Account"
            },
            {
                "value": [
                    "Product"
                ],
                "display": "Product"
            }
        ],
        "useGlobal": true,
        "datasets":[{"name":"opp"}],
        "type": "static",
        "selectMode": "singlerequired",
        "isGlobal": false
    }
}
```

The following example binds the array values to a SAQL query that requires the `"group by"` and `"foreach generate"` values to use single quotes. Therefore `single_quote()` converts `["Account-Name"]` to `'Account-Name'`.

```
{
    "step_Account_Name_1": {
        "isFacet": false,
        "query": {
            "pigql": "q = load \"opp\";\nq = group q by
                {{ single_quote(value(selection(step_StageName_3))) }};\nq =
                foreach q generate {{ single_quote(value(selection(step_StageName_3)))
                }} as {{ single_quote(value(selection(step_StageName_3)) }},
                sum('Amount') as 'sum_Amount', count() as 'count'",
            "groups": "{{ selection(step_StageName_3) }}",
            "measures": [
                [
                    "sum",
                    "Amount"
                ]
            ]
        },
        "visualizationParameters": {
            "visualizationType": "hbar"
        },
        "selectMode": "none",
        "useGlobal": true,
        "datasets":[{"name":"opp"}],
        "type": "aggregate",
        "isGlobal": false
    }
}
```

55

The resulting query is:

```
q = load "opp";\nq = group q by 'Account-Name';\nq =
             foreach q generate 'Account-Name' as 'Account-Name', sum('Amount') as
             'sum_Amount', count() as 'count'
```

no_quote()

The no_quote() operation is typically used in selection bindings in a SAQL step to correctly format the "order" line in a query. The no_quote() operation takes an array of values and converts double quotes and square brackets into no quotes. For example, ["desc"] converts to desc.

Consider the ["desc"] and ["asc"] array values that are specified in the following static step:

```
{
    "step_order": {
        "type": "static",
        "values": [
            {
                "display": "desc",
                "value": [
                    "desc"
                ]
            },
            {
                "display": "asc",
                "value": [
                    "asc"
                ]
            }
        ],
        "selectMode": "singlerequired"
    }
}
```

The following example binds the array values into a SAQL step:

```
q = order q by 'Amount' {{ no_quote(value(selection(step_order))) }}
```

The desc or asc value is inserted without any quotes:

```
q = order q by 'Amount' desc
```

field()

The field() operation creates a field for each object in an array.

Three field values are assigned to the "$" and "#" options in this static step (step_measure): "compact", "alias", and "proj":

```
{
    "step_measure": {
        "type": "static",
        "values": [
            {
```

```
                    "display": "$",
                    "value": [
                        {
                            "compact": [["sum", "Amount"]],
                            "alias": "sum_Amount",
                            "proj": "sum('Amount')"
                        }
                    ],
                    "display": "#",
                    "value": [
                        {
                            "compact": [["count", "*"]],
                            "alias": "count",
                            "proj": "count()"
                        }
                    ]
                }
            ],
            "selectMode": "singlerequired"
        }
}
```

After being assigned, each field value can be referenced in other step selection bindings by using the `field()` operation.

For example, when a dashboard user clicks **#** in the toggle selector that uses `step_measure`, the SAQL query in this aggregate step (`step_top_10`) references the `"proj"` field to insert a `count()` function, the `"alias"` field to insert `"count"` as a string, and the `"compact"` field to insert `[["count", "*"]]`.

```
{
  "step_top_10": {
    "type": "aggregate",
    "datasets":[{"name":"opp"}],
    "query": {
      "pigql":
      "q = load 'edgemarts/Opportunity/OpportunityEM';
      q = group q by 'Account_Name';
      q = foreach q generate
        'Account_Name' as 'Account_Name',
        {{ no_quote(value(field(selection(step_measure),'proj'))) }}
          as {{ single_quote(value(field(selection(step_measure), 'alias'))) }};
      q = order q by {{ single_quote(value(field(selection(step_measure), 'alias'))) }}
        {{ no_quote(value(field(selection(step_order), 'pigql'))) }};
      q = limit q {{ value(selection(step_limit)) }};",
      "groups": ["Account_Name"],
      "measures": "{{ value(field(selection(step_measure), 'compact')) }}",
      "order":
      [[ -1, { "ascending": "{{ value(field(selection(step_order), 'compact')) }}" } ]]
    },
  "isFacet": true
  }
}
```

GRID LAYOUTS

Wave inserts a `gridLayouts` section in your dashboard's JSON definition when you create a dashboard or save an existing dashboard with the flex dashboard designer. The `gridLayouts` section affects the layout of web browsers only on desktop and laptop computers, not mobile devices.

As you update the dashboard with the flex dashboard designer, the designer automatically updates the `gridLayouts` section of the dashboard JSON. You can also edit the JSON directly to set properties that aren't available in the user interface. To access the JSON editor, see View or Modify a Dashboard JSON File.

> 📝 **Note:** The `gridLayouts` section doesn't apply to mobile devices or to dashboards that are saved with the original designer. To configure how dashboards built with the original designer display on desktops, see Widgets. To configure the layout on mobile devices, see Use a Mobile Layout for Your Dashboard.

Grid Layouts Specification

The `gridLayouts` section allows you to customize how dashboards built with the flex dashboard designer display on desktops.

Grid Layouts Attribute Reference

Set attributes on widgets, rows, and cells to customize their height, width, and more.

Grid Layouts Specification

The `gridLayouts` section allows you to customize how dashboards built with the flex dashboard designer display on desktops.

In a dashboard's JSON file, the `gridLayouts` section is a child of the `state` section and a sibling of the `widgets` and `steps` sections. Here is an example of a typical `gridLayouts` section. The `widget_name` refers to a specific widget named in the `gridLayouts` section of the JSON file. It contains attributes that are defined in the Grid Layouts Attribute Reference.

```
"gridLayouts": [
    {
        "name": "desktop",
        "pages": [
            {
                "widgets": [
                    {
                        "colspan": 7,
                        "column": 0,
                        "name": "widget_name",
                        "row": 0,
                        "rowspan": 4
                    }
                ]
            }
        ],
        "selectors": [],
        "version": 1,
        "widgetStyle":
            {
                "borderEdges": ["all"],
```

```
                    "borderColor": "#44A2F5",
                    "backgroundColor": "#E2DCF2",
                    "borderWidth": 2
                }
            }
        }
]
```

Grid Layouts Attribute Reference

Set attributes on widgets, rows, and cells to customize their height, width, and more.
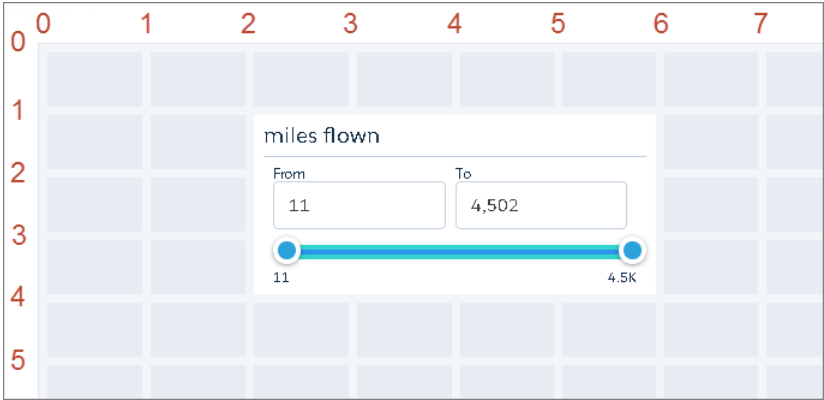
Widget Attributes

The widget attributes determine the height and width of each widget, and where it's placed on the dashboard. Because the flex dashboard designer uses a grid, you specify the attributes in terms of rows and columns. For example, you specify the number of columns to determine the width of a widget. You set the attributes for each widget under the `widgets` field under `gridLayouts`. The following JSON snippet shows an example of `gridLayouts` with widget attributes for the meafilter_1 widget.

```
"gridLayouts": [
    {
        "name": "desktop",
        "selectors": [],
        "version": 1,
        "widgetStyle": {},
        "pages": [
            {
                "widgets": [
                    {
                     "name": "meafilter_1",
                        "row": 1,
                        "column": 2,
                        "colspan": 4,
                        "rowspan": 3,
                        "widgetStyle":
                            {
                                "borderEdges": []
                            }
                    }
                ]
            }
        ]
    }
]
```

These widget properties place the widget as shown in the following graphic. Notice that each column and row start at 0, not 1.

A widget can have zero or more attributes.

Property Name	Details
name	**Type** String **Available for These Widgets** • All widgets **Description** Internal name of the widget. This name is used to reference the widget in the dashboard JSON.
column	**Type** Integer **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes. Value is determined based on the widget's placement. **Description** The column number where the widget starts. Column and row specify the top left corner of the widget. 📝 Note: If this widget is included in a container, these attributes are relative to the container widget.
row	**Type** Integer **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes. Value is determined based on the widget's placement. **Description** The row number where the widget starts. Column and row specify the top left corner of the widget.

Property Name	Details
colspan	**Type** Integer **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes. Value is determined based on the widget's placement. **Description** The number of columns that a widget spans—the width of the widget. If the dashboard doesn't have enough columns to accommodate the specified width, then columns are added to the dashboard.
rowspan	**Type** Integer **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes. Value is determined based on the widget's placement. **Description** The number of rows that a widget spans—the height of the widget. If the dashboard doesn't have enough rows to accomodate the specified height, then rows are added.
widgetStyle	**Type** List **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** No **Description** Specifies the border type, border color, and background color. See widgetStyle attributes.

Widget Style Attributes

The widget style attributes determine the border type, border color, and background color. You can specify these attributes at two levels. To set the default for all dashboard widgets, use the `widgetStyle` field under `gridLayouts`. To set a specific widget, use the `widgetStyle` field under `widgets`. This setting overrides the default settings.

The following JSON snippet shows `gridLayouts` with `widgetStyle` attributes for the meafilter_1 widget.

```
"gridLayouts": [
    {
        "name": "desktop",
        "selectors": [],
        "version": 1,
```

```
        "widgetStyle": {},
        "pages": [
            {
                "widgets": [
                    {
                        "name": "meafilter_1",
                        "row": 1,
                        "column": 2,
                        "colspan": 4,
                        "rowspan": 3,
                        "widgetStyle":
                            {
                                "borderEdges": ["all"],
                                "borderColor": "#44A2F5",
                                "backgroundColor": "#E2DCF2",
                                "borderWidth": 2
                            }
                    }
                ]
            }
        ]
}
]
```

The widgetStyle attributes specified in the previous JSON snippet format the widget's border and background color as shown here.

Property Name	Details
borderWidth	**Type** Integer **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes **Description** Width of the widget's border. Valid values are 1, 2, 4, and 8. Default is 2.

Property Name	Details
borderEdges	**Type** String **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes **Description** Specifies which edges of the widget have a border. Valid values are `left`, `right`, `top`, `bottom`, and `all`.
borderColor	**Type** String **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes **Description** Color of the widget's border. The default is `#FFFFFF`. If no border is specified, the widget has no border.
backgroundColor	**Type** String **Available for These Widgets** • All widgets **Exposed in the Dashboard Designer's User Interface** Yes **Description** Background color of the widget. The default is `#FFFFFF`.
bottomPadding	Reserved for future use.
leftPadding	Reserved for future use.
rightPadding	Reserved for future use.
topPadding	Reserved for future use.

LAYOUTS

Add a `layouts` section to your dashboard's JSON definition to customize its appearance on mobile devices.

There are two types of dashboard layouts for mobile devices:

Absolute (default)

If no `layouts` section is defined in your dashboard's JSON, then the dashboard's layout is absolute.

The absolute layout is optimized for display in a Web browser on a desktop or laptop computer.

Mobile

If a `layouts` section is present in your dashboard's JSON, then the dashboard's layout is mobile. This is true regardless if you use the original designer or flex dashboard designer to configure the dashboard.

The mobile layout lets you optimize the position, order, and size of the widgets in your dashboard for display on mobile devices. This layout is made up of rows, columns, cells, and pages. Each cell in the grid can contain zero or more widgets. The number of rows, columns, and cells in your mobile layout depend on the number of widgets and the number of pages.

A dashboard with an absolute layout looks great in a Web browser:
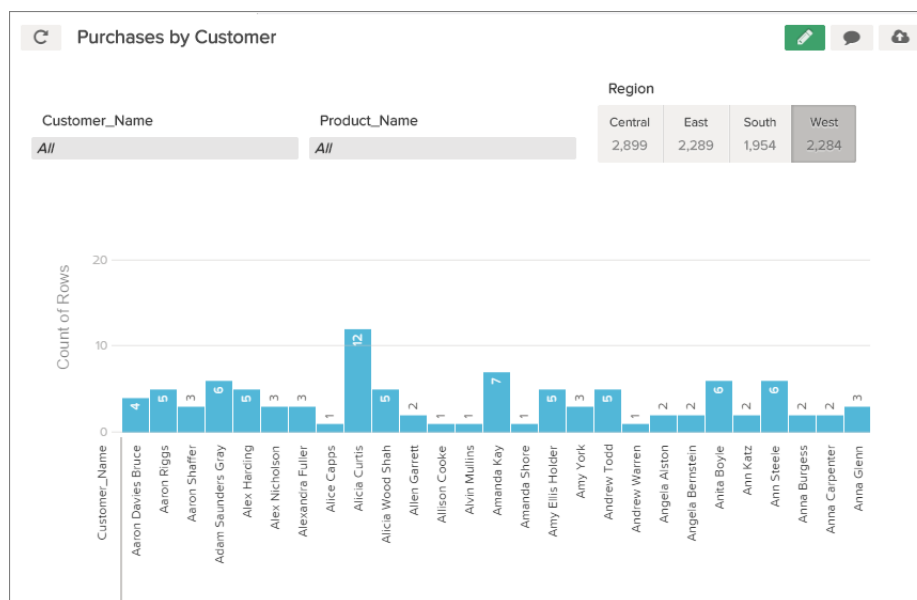
The same dashboard with an absolute layout might not render well on a smart phone:
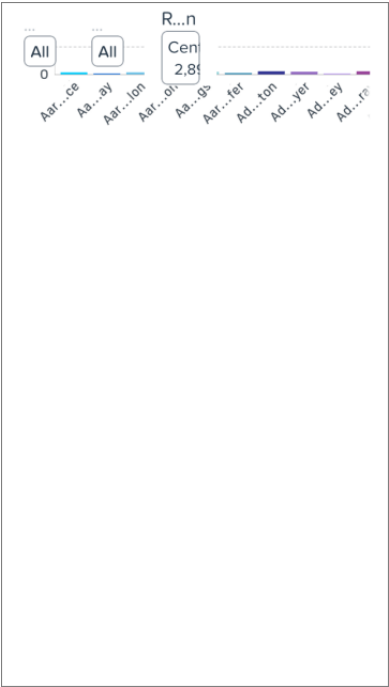
By using a mobile layout with two pages, the dashboard renders perfectly on a smart phone:
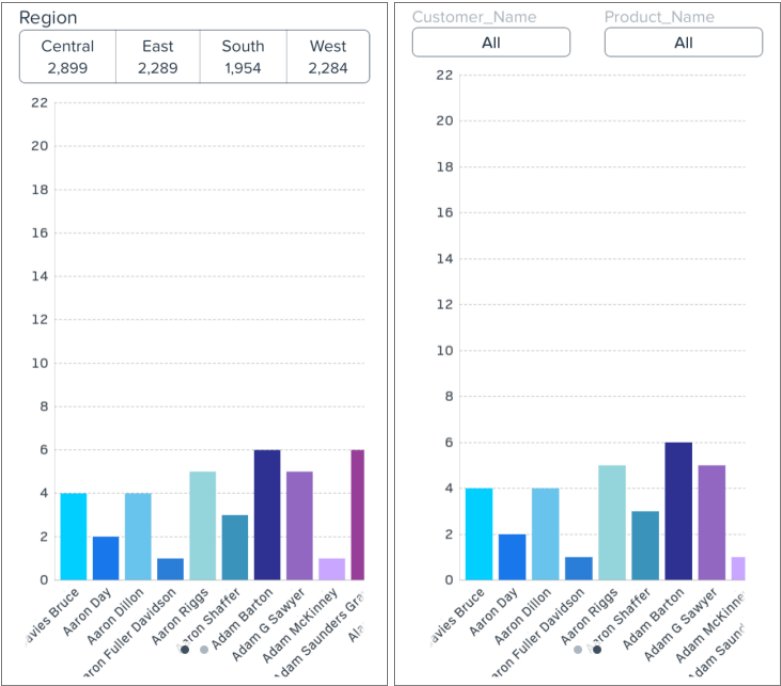
Use a Mobile Layout for Your Dashboard

Use a mobile layout to customize your dashboard's appearance on mobile devices.

Understanding Column, Row, and Cell Sizing in Mobile Layouts

Widgets size, row size, and the number of columns are determined dynamically, but can also be specified in the JSON.

Layouts Specification

The `layouts` section is used to customize how dashboards display on mobile devices.

Layouts Attribute Reference

Set attributes on widgets, rows, and cells to customize their height, width, padding, and more.

Use a Mobile Layout for Your Dashboard

Use a mobile layout to customize your dashboard's appearance on mobile devices.

In a dashboard's JSON file, the `layouts` section is a child of the `state` section and a sibling of the `widgets` and `steps` sections.

1. From the open dashboard, press CTRL+E for PC or CMD+E for Mac. This opens expert editor mode. For more information, see View or Modify a Dashboard JSON File.

2. Add a `layouts` section to your dashboard's JSON.

For example, this `layouts` section defines a mobile layout with two pages, two rows of widgets on each page. The first page has 1 widget on each row. The second page has two widgets on the first row, and one widget on the second row.

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "buttongroup_2",
          "chart_1"
        ]
      },
      {
        "rows": [
          "dimfilter_1 | dimfilter_3",
          "chart_1"
        ]
      }
    ],
    "version": 1
  }
```

3. Optionally, customize the layout of your dashboard by setting attributes for each widget and row.

For example, the *layouts* from step two can be updated to include widget and row attributes. The first row on the first page has a row height of 300 pixels. The chart widget on the second page has a width of 2 columns.

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "buttongroup_2 | row:{height=300}",
          "chart_1"
        ]
      },
      {
```

```
      "rows": [
        "dimfilter_1 | dimfilter_3",
        "chart_1 {colspan=2}"
      ]
    }
  ],
  "version": 1
}
```

4. Optionally, set device-specific and orientation-specific layouts for your dashboard. For available device and orientation options, see Layouts Options in the Layouts Specification guide.

   For example, the *layouts* from step three can be updated to use only one page when viewed on an iPad in landscape mode:

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "buttongroup_2 | row:{height=300}",
          "chart_1"
        ]
      },
      {
        "rows": [
          "dimfilter_1 | dimfilter_3",
          "chart_1 {colspan=2}"
        ]
      }
    ],
    "version": 1
  },
  {
    "device": "ipad",
    "orientation": "landscape",
    "pages": [
      {
        "rows": [
          "dimfilter_1 | dimfilter_3 | buttongroup_2",
          "chart_1 {colspan=3}"
        ]
      }
    ],
    "version": 1
  }
```

5. Click **Switch to Runtime**, and then save your updated dashboard.

6. Test your dashboard's new mobile layout by viewing the dashboard on a mobile device.

SEE ALSO:

Layouts Specification

Layouts Attribute Reference

Layouts Specification

Understanding Column, Row, and Cell Sizing in Mobile Layouts

Widgets size, row size, and the number of columns are determined dynamically, but can also be specified in the JSON.

How Column Number and Size Are Set

The number of columns in your mobile layout is equivalent to the number of widgets in your rows. If there are three widgets in each row, then the dashboard has three columns. If your mobile layout has two rows with four widgets in row one and five widgets in row two, then the dashboard has five columns. If the `colspan` attribute specifies a number of columns greater than the number of widgets in any row, then the dashboard adds columns to accommodate the `colspan` attribute.

For example, a dashboard with this `layouts` section has three columns on the first page and two columns on the second page:

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "buttongroup_2",
          "chart_1 {colspan=3}"
        ]
      },
      {
        "rows": [
          "dimfilter_1 | dimfilter_3",
          "chart_1"
        ]
      }
    ],
    "version": 1
  }
```

Remember these tips when determining how many columns are in your mobile layout:

- All columns have the same width. If your dashboard has four columns, then each column is half the width of a dashboard with two columns.
- Each page of a dashboard independently determines how many columns appear. For example, a dashboard can have three columns on page one, and four columns and page two.
- Every dashboard has at least one column.
- There is no limit to the number of columns that a dashboard can have. If you add too many columns, then column width could become impracticably small. Remember to test your layout for usability!

How Row Number and Height Are Set

For each row, here's how height is calculated:

- If a row height is set using the `height` attribute, then the row's height is equal to the specified value.
- If one or more widgets in the row has a preferred height, then the row's height is equal to that of whichever preferred height is tallest.
- If there is no `height` attribute and none of the row's widgets have a preferred height, then the row's height dynamically grows to occupy the available space. If multiple rows grow dynamically, then their heights are equal to one another. For example, if there are 200 pixels of available space, and two rows with dynamically set heights, then each row has a height of 100 pixels.

How Widgets Are Sized

Some widgets have absolute sizes, and some scale dynamically.

Widget	Has a Fixed Width?	Has a Fixed Height?	Width Scaling Behavior	Height Scaling Behavior
Link	Yes	Yes	Don't scale	Don't scale
Text	No	If one line long, yes. If more than one line long, no.	Scale to fit text	Scale to fit text
Pillbox	No	Yes	Scale	Don't scale
Box	No	No	Scale	Scale
Chart	No	No	Scale	Scale
List selector	No	Yes	Scale	Don't scale
Range selector	No	Yes	Scale	Don't scale
Number	No	Yes	Scale	Don't scale

Layouts Specification

The `layouts` section is used to customize how dashboards display on mobile devices.

In a dashboard's JSON file, the `layouts` section is a child of the `state` section and a sibling of the `widgets` and `steps` sections. Here is an example of a typical `layouts` section:

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "widget_name_1",
          "widget_name_2"
        ]
```

```
    },
    {
      "rows": [
        "widget_name_3 | widget_name_4",
        "widget_name_2 {attribute=2}"
      ]
    }
  ],
  "version": 1
},
{
  "device": "ipad",
  "orientation": "landscape",
  "pages": [
    {
      "rows": [
        "widget_name_1 | widget_name_3 | widget_name_4 | row: {attribute=300}",
        "widget_name_2 {widget_name=3}"
      ]
    }
  ],
  "version": 1
}
```

In the prior example, `widget_name` refers to a specific widget named in the `widgets` section of the JSON file. `Attribute` refers to one of the attributes listed in the Layouts Attribute Reference. The pipe character (|) is the delimiter for cells. A cell can contain multiple widgets separated by a comma (,). Rows are delimited by a comma (,) outside the quoted string (each quoted string is a single row).

Simple Layouts Section

Here's a simple `layouts` section that has four widgets on four rows in a single column on a single page:

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "buttongroup_1",
          "dimfilter_1",
          "dimfilter_2",
          "chart_1"
        ]
      }
    ]
    "version": 1
  }
```

Complex Layouts Section

A more complex `layouts` section can be used to set device-specific and orientation-specific display rules. The following `layouts` section lays out the dashboard's widgets on two pages. The first page's first row has a height of 300 pixels. The second page has two

rows and two columns. One of the cells in the first row contains two widgets. One of the box widgets has three attributes set. The chart widget spans two columns. If the dashboard is viewed on an iPad in landscape mode, then only one page with two rows is shown. The first row has three widgets and the second row has one widget that spans three columns.

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "buttongroup_2 | row: {height=300}",
          "chart_1"
        ]
      },
      {
        "rows": [
          "dimfilter_1, box_1 {colspan=2, rowspan=2, zIndex=-1, vpad=5, hpad=5} |
            dimfilter_2","chart_1 {colspan=2}"
        ]
      }
    ],
    "version": 1
  },
  {
    "device": "ipad",
    "orientation": "landscape",
    "pages": [
      {
        "rows": [
         "dimfilter_1, box_1 {colspan=2, rowspan=3, zIndex=-1, vpad=5, hpad=5} | dimfilter_2
|
            buttongroup_2",
          "chart_1 {colspan=3}"
        ]
      }
    ],
    "version": 1
  }
```

Layouts Options

The previous example shows a layout specifically for an iPad in landscape mode (`device:ipad, orientation:landscape`). Layout device and orientation choices are as follows:

- `"device":"default"`: For layouts not targeted to any specific device or orientation.
- `"device":"ipad", "orientation":"portrait"`: For Apple iPad in portrait mode.
- `"device":"ipad, "orientation":"landscape"`: For Apple iPad in landscape mode.
- `"device":"ipad"`: For Apple iPad in either portrait or landscape mode.
- `"device":"iphone"`: For Apple iPhone; portrait mode is implied.
- `"device":"external"`: For displaying on an external device, for example if device is connected via HDMI cable to a projector or display. To use `external` layout, select Presentation Mode in Settings.
- `"device":"applewatch"`: For Apple Watch. Supports only a single, scrolling page.

- "orientation":"portrait": For either iPhone or iPad in portrait mode.
- "orientation":"landscape": For iPad in landscape mode.

> 📝 **Note:** If the app is viewed on Apple Watch and "device":"applewatch" layout is not present, the app first tries to reformat the first page of the "device":"iphone" layout. If "device":"iphone" is not present, it then attempts to use the first page of the "device":"default" layout.

> 📝 **Note:** If the app is viewed on an external device and "device":"external" layout is not present, the app first tries to use the first page of the "device":"ipad" "orientation":"landscape". If "device":"ipad" "orientation":"landscape" is not present, it then attempts to use the first page of the "device":"default" layout.

Layout Autoformatting

If layouts is not specified, Wave uses autoformatting to present the dashboard, which takes a best guess about the appropriate layout to use. Note the following about layout autoformatting:

- With AppleWatch, autoformat uses the first page of the default layout and converts it to a single column.
- With an external device, autoformat supports only a single, unscrollable page and attempts to fit all the dashboard contents on the external display.
- Autoformat supports a limited number of columns on each device, as shown in the table.

Device	Maximum columns supported by autoformatting
Apple Watch	One
Apple iPhone	Two
Apple iPad	Four

Autoformatting is enabled by default. To disable autoformatting, for example for a carefully designed dashboard that cannot use a mobile layout, add an empty pages array under the layouts array, which looks like this:

```
"layouts": [
  {
    "pages": [
      {
      }
    ]
  }
```

SEE ALSO:

Layouts Attribute Reference

Set attributes on widgets, rows, and cells to customize their height, width, padding, and more.

Widget Attributes

These attributes can be set on widgets. Each widget can have zero or more attributes.

Property Name	Details

colspan	**Type**
	Integer
	
	Available for These Widgets
	
	- All widgets
	
	Description
	The number of columns that a widget spans—the width of the widget. If the dashboard doesn't have enough columns to accommodate the specified width, then columns are added to the dashboard.
	
	Example
	In this example, the widget named "chart_1" spans 3 columns:
	
	<pre>"layouts": [
	  {
	    "device": "default",
	    "pages": [
	      {
	        "rows": [
	          "dimfilter_1 | dimfilter_2 | dimfilter_3",
	          "chart_1 {colspan=3}"
	        ]
	      }
	    ]
	    "version": 1
	  }</pre>

rowspan	**Type**
	Integer
	
	Available for These Widgets
	
	- All widgets
	
	Description
	The number of rows that a widget spans—the height of the widget. If the dashboard doesn't have enough rows to accomodate the specified height, then rows are added.
	
	Example
	In this example, the widget named "dimfilter1_1" spans 2 rows:
	
	<pre>"layouts": [
	  {
	    "device": "default",</pre>

Property Name	Details

```
    "pages": [
      {
        "rows": [
          "dimfilter_1 {rowspan=2} | dimfilter_2",
          "chart_1"
        ]
      }
    ]
    "version": 1
  }
```

zIndex

Type
Integer

Available for These Widgets

- All widgets

Description

The position of a widget relative to other widgets in the dashboard. `zIndex` specifies whether a widget is in front of or behind another widget. A smaller `zIndex` means that a widget appears further behind other widgets with larger `zIndex` values.

The default value of `zIndex` is 0.

Example

In this example, the widget named "`box_1`" appears behind the widget named "`number_1`":

```
"layouts": [
  {
    "device": "default",
    "pages": [
      {
        "rows": [
          "box_1 {zIndex=1}, number_1 {zIndex=2} | chart_1"
        ]
      }
    ]
    "version": 1
  }
```

vpad

Type
Integer

Available for These Widgets

- All widgets

Description

The padding added to the top and bottom sides of the widget's cell in pixels. If `vpad` equals 10, then 10 pixels are added to the top of the cell and 10 pixels are added to the bottom.

The default value of `vpad` is 0.

Property Name	Details

	Example
	In this example, the cell containing widget named "`dimfilter_1`" has 5 pixels of padding on its top and bottom sides:
	
	``json
	"layouts": [
	  {
	    "device": "default",
	    "pages": [
	      {
	        "rows": [
	          "dimfilter_1 {vpad=5}"
	        ]
	      }
	    ]
	    "version": 1
	  }
	``

`hpad`	**Type**
	Integer
	
	Available for These Widgets
	• All widgets
	
	Description
	The padding added to the left and right sides of the widget's cell in pixels. If `hpad` equals 10, then 10 pixels are added to the left side of the cell and 10 pixels are added to the right side. A negative value can be assigned to
	
	The default value of `hpad` is 0.
	
	Example
	In this example, the cell containing widget named "`dimfilter_1`" has 5 pixels of padding on its top and bottom sides:
	
	``json
	"layouts": [
	  {
	    "device": "default",
	    "pages": [
	      {
	        "rows": [
	          "dimfilter_1 {hpad=5}"
	        ]
	      }
	    ]
	    "version": 1
	  }
	``

`vAxisWidth`	**Type** Integer **Available for These Widgets** • chart

Property Name	Details

	Description
	The size of a chart widget's x-axis in pixels. Use `vAxisWidth` to align multiple chart widgets.
	
	Example
	In this example, the widget named "`chart_1`" has an x-axis that is 250 pixels wide:
	
	<pre>"layouts": [
	  {
	    "device": "default",
	    "pages": [
	      {
	        "rows": [
	          "chart_1 {vAxisWidth=250}"
	        ]
	      }
	    ]
	    "version": 1
	  }</pre>

`hAxisHeight`	**Type**
	Integer
	
	Available for These Widgets
	    `chart` **Description**
	The size of a chart widget's y-axis in pixels. Use `hAxisHeight` to align multiple chart widgets.
	
	Example
	In this example, the widget named "`chart_1`" has a y-axis that is 250 pixels tall:
	
	<pre>"layouts": [
	  {
	    "device": "default",
	    "pages": [
	      {
	        "rows": [
	          "chart_1 {hAxisHeight=250}"
	        ]
	      }
	    ]
	    "version": 1
	  }</pre>

Row Attributes

These attributes can be set on rows.

Property Name	Details

height	**Description**
	If `height` is set to a number, then `height` is the height of a row in pixels.
	
	If `height` is set to *preferred*, then the row's height is equal to the largest height
	
	Example
	In this example, the first row's height is 300 pixels. The second row's height is equal to the height of its tallest widget:
	
	<pre>"layouts": [
	  {
	    "device": "default",
	    "pages": [
	      {
	        "rows": [
	          "chart_1 {colspan=3} | row:{height=300}",
	          "dimfilter_1 | buttongroup_1 | number_1 |
	row:{height=preferred}"
	        ]
	      }
	    ]
	    "version": 1
	  }</pre>

SEE ALSO:

Use a Mobile Layout for Your Dashboard

Layouts Specification

CHART TYPES FOR WIDGETS

You can display a widget as a chart to visualize your data. Wave provides different types of charts to represent and highlight data in different ways. This section describes some of them.

Waterfall Chart Type Example

Sample JSON and chart widget for `waterfall` chart.

Stacked Waterfall Chart Type Example

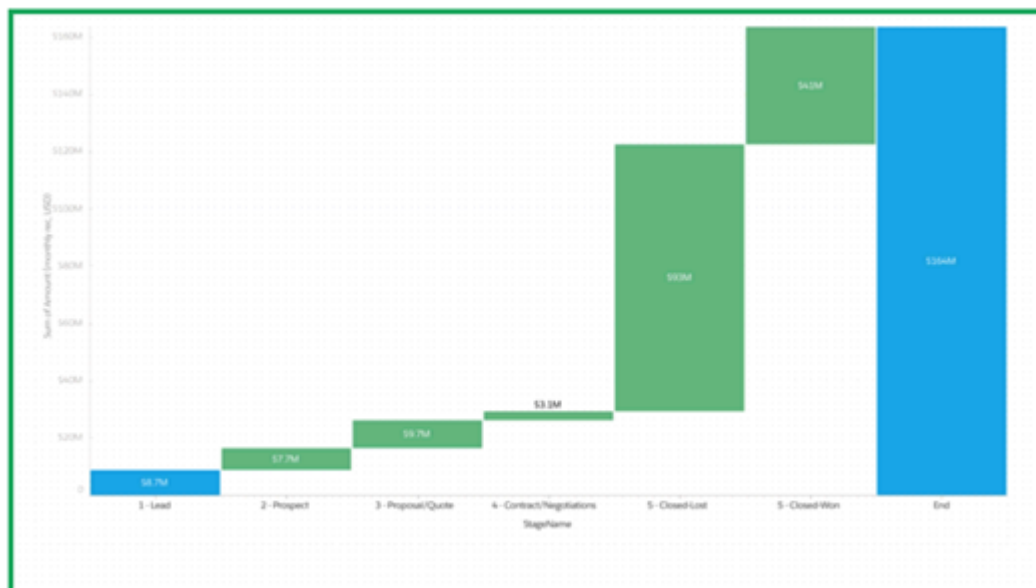Sample JSON and chart widget for `stackwaterfall` chart.

Combo Chart Type Example

Sample JSON for `combo` chart.

Waterfall Chart Type Example

Sample JSON and chart widget for `waterfall` chart.
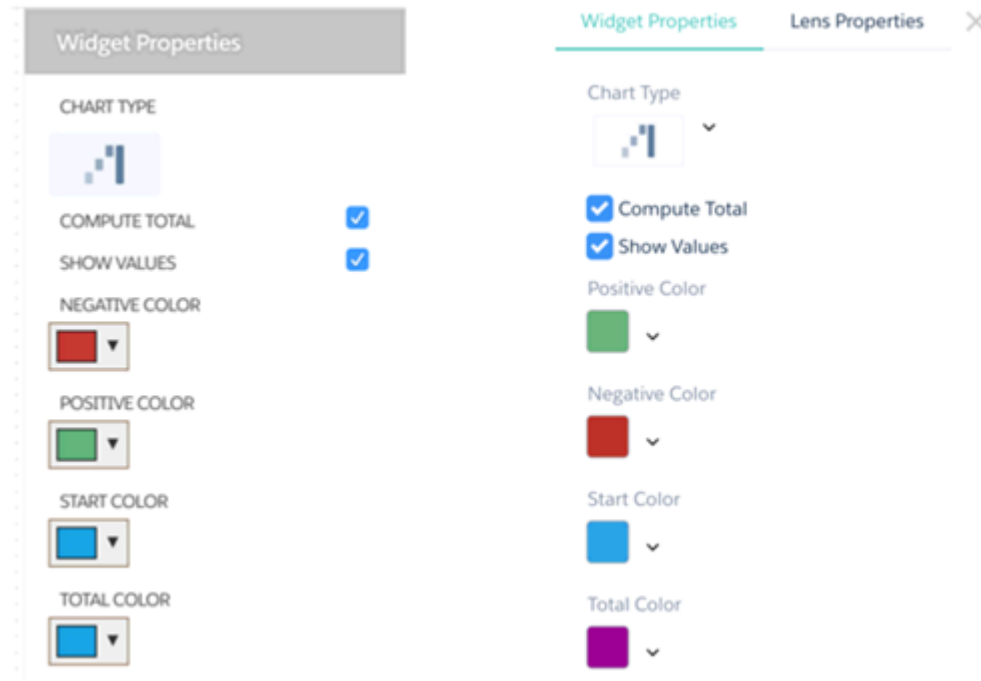

Sample JSON

```
"ng1":
  {
 "type": "chart",
 "position": {
   "x": 0,
   "y": 0,
   "w": "380",
   "h": "320"
```
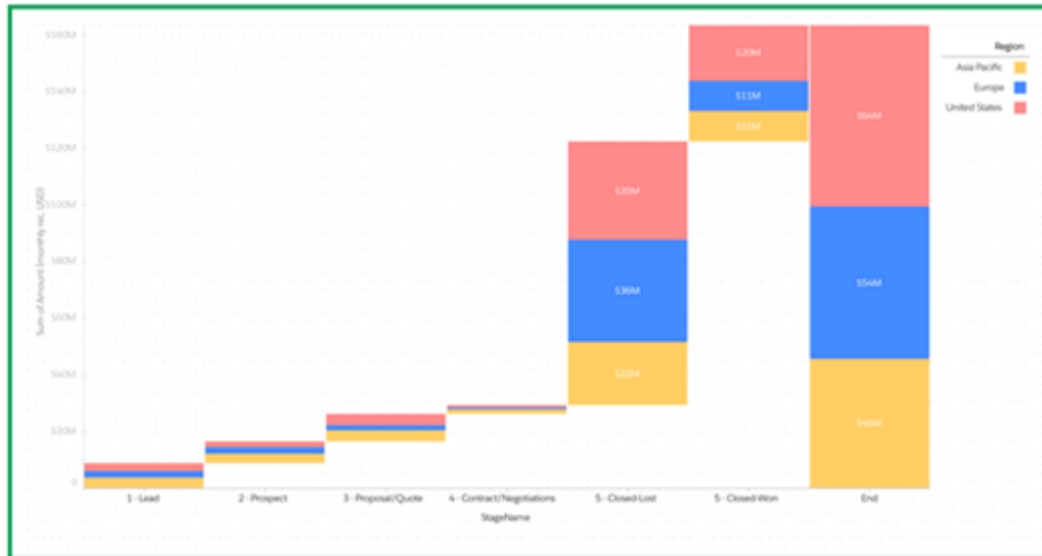
```
  },
  "parameters": {
   "step": "step_one_dim",
   "totalColor": "rgb(163, 24, 147)",
   "visualizationType": "waterfall"
  }
}
```

Chart Widget

Stacked Waterfall Chart Type Example

Sample JSON and chart widget for `stackwaterfall` chart.

Sample JSON

```
"ng2":
{
 "type": "chart",
 "position": {
  "x": 450,
  "y": 0,
  "w": "440",
  "h": "320"
 },
 "parameters": {
  "step": "step_two_dims",
  "visualizationType": "stackwaterfall",
  "computeTotal": false,
              "showValues": false,
              "legend": true
 }
}
```
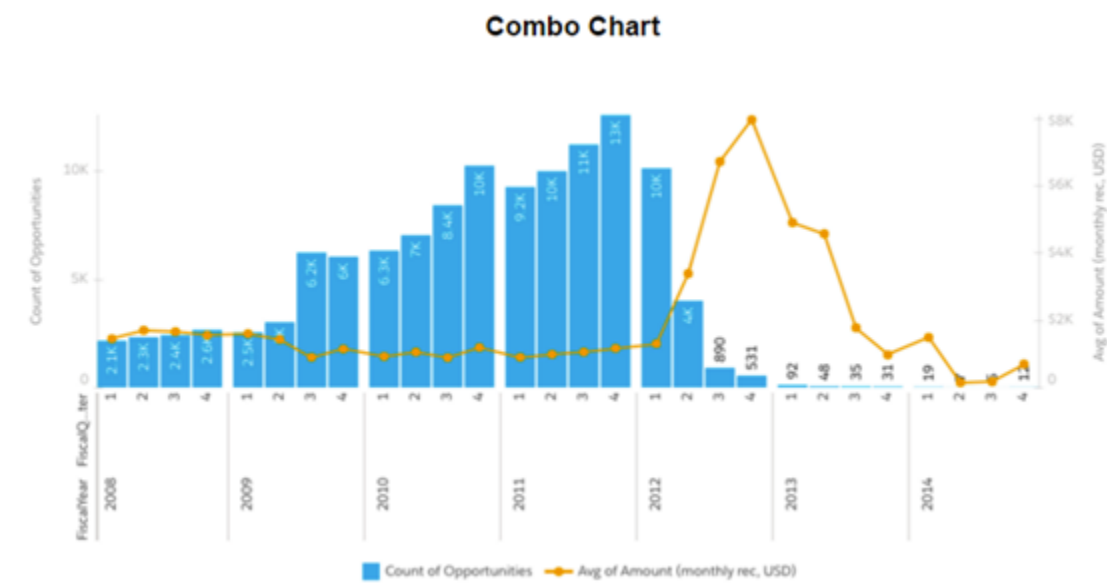
Chart Widget

Combo Chart Type Example

Sample JSON for `combo chart`.

Sample JSON

```
"chart_19":
{
 "type": "chart",
 "position": {
  "zIndex": 18,
  "x": 0,
  "y": 10,
  "w": "990",
  "h": "430"
```

```
  },
  "parameters": {
   "step": "FiscalYear_FiscalQuarter_6",
   "visualizationType": "combo",
   "dualAxis": true
  }
},
```

```
  },
  "parameters": {
   "step": "FiscalYear_FiscalQuarter_6",
```